Information and Software Technology 55 (2013) 1925–1947

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A formal framework for software product lines ☆

César Andrés, Carlos Camacho, Luis Llana *

Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Madrid, Spain

ARTICLE INFO

Article history:
Received 24 May 2012
Received in revised form 19 May 2013
Accepted 20 May 2013
Available online 31 May 2013

Keywords:
Formal methods
Software product lines
Feature models

ABSTRACT

Context: A Software Product Line is a set of software systems that are built from a common set of features. These systems are developed in a prescribed way and they can be adapted to fit the needs of customers. Feature models specify the properties of the systems that are meaningful to customers. A semantics that models the feature level has the potential to support the automatic analysis of entire software product lines.
Objective: The objective of this paper is to define a formal framework for Software Product Lines. This framework needs to be general enough to provide a formal semantics for existing frameworks like FODA (Feature Oriented Domain Analysis), but also to be easily adaptable to new problems.
Method: We define an algebraic language, called SPLA, to describe Software Product Lines. We provide the semantics for the algebra in three different ways. The approach followed to give the semantics is inspired by the semantics of process algebras. First we define an operational semantics, next a denotational semantics, and finally an axiomatic semantics. We also have defined a representation of the algebra into propositional logic.
Results: We prove that the three semantics are equivalent. We also show how FODA diagrams can be automatically translated into SPLA. Furthermore, we have developed our tool, called AT, that implements the formal framework presented in this paper. This tool uses a SAT-solver to check the satisfiability of an SPL.
Conclusion: This paper defines a general formal framework for software product lines. We have defined three different semantics that are equivalent; this means that depending on the context we can choose the most convenient approach: operational, denotational or axiomatic. The framework is flexible enough because it is closely related to process algebras. Process algebras are a well-known paradigm for which many extensions have been defined.

1. Introduction

Software Product Lines [1,2], in short SPLs, constitute a paradigm for which industrial production techniques are adapted and applied to software development. In contrast to classical techniques, where each company develops its own software product, SPLs define generic software products, enabling mass customization [3]. Generally speaking, using SPLs provide a systematic and disciplined approach to developing software. It covers all aspects of the software production cycle and requires expertise in data management, design, algorithm paradigms, programming languages, and human–computer interfaces.

When developing SPLs, it is necessary to apply sound engineering principles in order to obtain economically reliable and efficient software. *Formal methods* [4–9] are useful for this task. A formal method is a set of mathematical techniques that allows automated design, specification, development and verification of software systems. For this process to work properly, a well defined formalism must exist. There are many formalisms to represent SPLs [10–13]. We focus on one of the most widely approaches: *Feature models* [10,13].

A feature model is a compact representation of all the products of an SPL in terms of commonality and variability. Generally these features are related using a tree-like diagram. A variation point is a place where a decision can be made to determine if none, one, or more features can be selected to be part of the final product. For instance, in these models we can represent the following property:

There exists a product with features A *and* C.

In addition, it is easy to represent constraints over the features in feature models. For instance, we could represent the following property:

In any valid product, if feature C *is included then features* A *and* B *must also be included.*

☆ Research partially supported by the Spanish MEC project TIN2009-14312-C02-01 and TIN2012-36812-C02-01.

* Corresponding author. Tel.: +34 913944527.
E-mail addresses: rasec.andres@gmail.com (C. Andrés), carloscamachoucv@gmail.com (C. Camacho), llana@ucm.es (L. Llana).

0950-5849/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2013.05.005

Feature Oriented Domain Analysis [10], in short FODA, is a feature model to represent SPLs. This model allows us to graphically represent *features* and their *relationships*, in order to define *products* in an SPL. The graphical structure of a FODA model is represented by a FODA *Diagram*. A FODA Diagram is essentially an intuitive and easy to understand graph where there is relevant information about the features. This diagram has two different elements: the set of nodes and the set of arcs. The former represents the features of the SPL. The latter represents the relationships and the constraints of the SPL. We introduce the basic components of a FODA diagram in Fig. 1. With these elements we can model complex SPLs.

For instance, let us look at the FODA Diagrams in Fig. 2. The Examples **a** and **b** show two SPLs with possibly two possible features A and B. With respect to **a**, the feature A will appear in all valid products of this SPL, while B is optional. Therefore, the valid set of products of this FODA Diagram is one product with A and one product with features A and B. In **b** both features are mandatory, i.e. any product generated from this SPL will contain features A and B.

Example **c**, represents an SPL with a *choose-one* operator. There are three different features: A, B and C in this diagram. Any valid product of **c** will contain A and one of these features B or C. The *conjunction* operator is shown in **d** and **e**. In both examples two branches leave feature A. On the one hand, the branches in Example **d** are mandatory. This means that there is only one product derived from this diagram: the one that contains features A, B, and C. On the other hand, one branch in Example **e** is optional and the other is mandatory. This means that there are two products derived from this diagram: one with features A and C, and one with features A, B, and C.

Finally, more complex properties appear in Examples **f** and **g**. In these diagrams there are tree constraints combined with optional features. In **f**, there is an *exclusion* constraint: If B is included in a product then feature C cannot appear in the same product. In **g**, there is a *require* constraint: If B is included then C must also be included.

Although FODA Diagrams are very intuitive, sometimes it can be hard to analyze all the restrictions and the relationships between features. In order to make a formal analysis we have to provide a *formal semantics* for the diagrams. To obtain a formal semantics for SPLs, first we need a formal language. In this paper we define a formal language called SPLA. As we will see in Section 3, all FODA Diagrams can be automatically translated into SPLA.

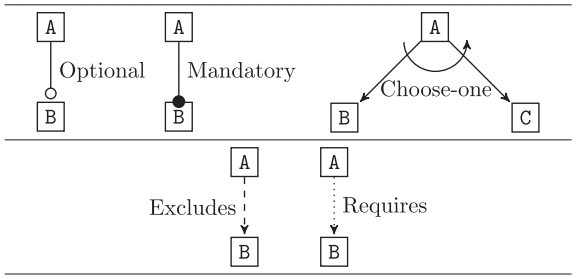

Fig. 1. FODA diagram representation.

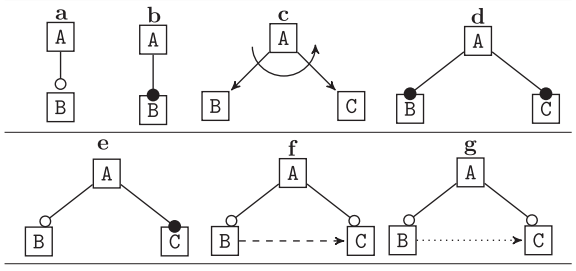

Fig. 2. Examples of FODA diagrams.

After presenting SPLA, we need to introduce the formal semantics of this algebra. There is previous work on formalizing FODA and feature models [30,22,31–33,23,24] that we briefly review in the next section. The approach we follow in this paper is inspired by classical process algebras [4,6,5]. We define three different semantics for the language. First we introduce an operational semantics whose computations give the products of an SPL. Next we define a denotational semantics that is less intuitive but easier to implement. Finally we have defined an axiomatic semantics. We prove that all three semantics are equivalent to each other.

In addition to presenting the formal framework, we have developed a tool called AT. This tool is an implementation of the formal semantics presented in this paper. Using AT it is possible to check properties such as:

- *Can this SPL produce a valid product?*
- *How many valid products can we build within this SPL?*
- *Given an SPL diagram, can we generate an equivalent SPL diagram with fewer restrictions than the first one?*

This tool is completely implemented in JAVA. This tool has a module to check the satisfiability of an SPL diagram. We carry out some experiment using diagrams obtained from the random SPL diagram generator Betty. These experiments shows the scalability of AT. We have checked satisfiability of diagrams with 13,000 features; such diagrams are relatively large given the state of the art.

In this paper we present a syntactic and semantic framework that formalizes FODA-like diagrams. First we present a syntax with the basic operators presented in a FODA-like diagram. Next we define an operational semantics. This semantics is intuitive and it captures the notions of the operators. After the operational semantics, we give the denotational semantics. This semantics is more appropriate for obtaining the products of an SPL. We prove that both semantics are equivalent. We also define an axiomatic semantics. As far as we know, this semantics does not appear in any of the previous frameworks. We prove that this semantics is sound and complete with respect to the previous ones. Inspired by recent works in process algebras, we also give a way to represent the terms of the algebra into propositional logic. Finally we have implemented a tool that supports our framework. The tool is split into two modules. One module deals with the denotational and axiomatic semantics while the other deals with the representation into propositional logic. The second module uses SAT-solver to check the satisfiability of an SPL.

This paper tries to show that SPLs can benefit from the process algebra community mainly because process algebras have been studied from many points of view. For instance, there have been numerous proposals to incorporate non-functional aspects such as time and probabilities. In particular, we are currently working in the following aspects. First, we are studying how to introduce, the notions of costs and time. In this context, it is also important the order in which products are elaborated, and therefore, sequences instead of sets have to be used. Second, we would like to work with models that indicate the probability of a product. This can be applied, for example, in software testing, so that we can add more resources to test the products with higher probabilities. Moreover, our semantic approach, based on alternative semantics that are shown to be equivalent, can be used in further extensions of both the formalism used in this paper and other formalisms of similar nature.

The paper is structured as follows. In Section 2 we review different formal models used to represent SPLs and compare other algebraic approaches with the one presented in this paper. In Section 3 we show the full syntax of our algebra and how any FODA Diagram is translated into the terms of SPLA. In Sections 4–6 the operational, denotational and the axiomatic semantics (respectively) of SPLA are presented. Next, in Section 8 we show the applicability of this approach in a complete study of a video streaming system. In Section 9 we present some features of our tool AT, that implements our formal framework. Finally, we conclude this paper in Section 10, by presenting some conclusions and presenting possible lines of future work. In addition to these sections, there is an Appendix with all the proofs of the results of this paper.

2. Related work

In this section we present the most usual formal frameworks used to model SPLs. Let us note that formal methods use both a formal *syntax* and a formal *semantics*. The former is used to write a precise specification of a system. With the latter a precise meaning is given for each description in the language.

Several proposals exist to formally describe SPLs. We have classified these formal frameworks into two categories. On the one hand, there are proposals that adapt well-known formal frameworks, like transition systems [14–21], to represent SPLs. On the other hand, there are proposals that give a semantics to originally informal frameworks dealing with SPL, such as Feature Models [22–24].

Next we are going to discuss some frameworks that are in the first category. First we describe frameworks [14–18] that deal with adaptations of transition systems. The most widely used model is the *Modal Transition System*, in short MTS. An MTS is an extension of a *Labelled Transition System*. In MTSs, there are two types of transitions: *obligatory* and *optional*, representing transitions that are always performed and transitions that may be included or discarded respectively. Furthermore, they present different notions of conformance between sets of products. Let us note that the original MTS models did not have the capability to define constraints between features. This problem was overcome with the Extended Modal Transition Systems [15] and by an associated set of logical formulae [19,21]. There are other studies that have been able to extend the functionality of MTSs. In particular, the authors in [16] define the Modal I/O Automata to represent the behavior of SPLs. Finally, it is worth mentioning another approach that models SPLs based on Petri Nets [25].

There are other frameworks that follow an algebraic approach [26–28]. In mathematical terms an *algebra* consists of a set of symbols denoting *values* of some type, and *operations* on the values. In [26,27] the authors define the SPLs as idempotent semirings where the values are the features. The denotational semantics presented in this paper is closely related to this model. Gruler et al. [28] present an algebra based on the classical CCS [4] process algebra called PL-CCS. This framework is also closely related to ours. Later we describe how our approach relates to theirs. Finally, in [29] SPLs are represented as logical expressions.

With regards to the second category, we can mention the following [30,22,31–33,23,24]. In these frameworks authors provide a semantics for existing feature models [34,10,35,36] such as FODA and its extension RSEB. Benavides et al. [23] make an automated analysis of feature models. In [22] the author uses the model defined in [29] to model FODA. In [30] and its extension [32] the authors define a semantics for FODA based on what they call a *tree feature diagram*. [24] deserves special mention since the authors present a general framework where they can define semantics for all graph-like diagrams like FODA. The authors in [33] translate a feature diagram into propositional logic. In this way they can verify the consistency of models of at least 10,000 features.

In our framework, we present a process algebra to represent SPLs. In this sense our approach is a new formalism to represent SPLs, so it lies in the first category. One of the main objectives of our proposal is to define a general framework that can be used to provide FODA, or other graph-like diagrams, with a *formal semantics* that removes any ambiguity or lack of precision as in [32,24]. Therefore our approach also lies in the second category. The approach given by our semantics is similar to the one in [28], but our intention has not been to extend an existing process algebra but to define an algebra to represent graph-like diagrams with cross tree constraints, like FODA. In this sense our approach is simpler and the semantics is also simpler. In particular, we do not need the fixed point theory because there is no recursion in the graph-like diagrams we study. However, it would have been possible to include a recursion-like operator. Since our semantics is based on traces, it is likely that we can define a complete partial order in our semantics so that all the operators are continuous. The algebraic approach in [26,27] is similar to our denotational semantics. They present a semiring with two operators that correspond to our *choose-one* and *conjunction* operators, and two constants that correspond to our $\swarrow$ and nil constants. They present neither an operational nor an axiomatic semantics. Let us note that one advantage of our approach is that it is based on the well-known formalism of process algebras. This formalism is very flexible and many extensions of process algebras have been studied to incorporate non-functional features like time and probabilities. we anticipate that this will allow us to incorporate new characteristics in the future (see Section 10).

To conclude this section we would like to mention that SAT-solvers have been used recently in the context of process algebras [37–39].

3. FODA algebra

In this section we present the syntax of our algebra and how FODA Diagrams are translated into this algebra. First, in sub section 3.1 we define the syntax of the algebra and we explain the intuitive meaning of the operators. Next, in Section 3.2 we introduce the translation of a FODA Diagram into our syntax.

3.1. Syntax of FODA

The *syntax* concerns the principles and rules for constructing *terms*. We define the language SPLA by means of an Extended BNF expression. In order to define the syntax, we need to fix the set of *features*. From now on $\mathcal{F}$ denotes a finite set of features and A, B, C, etc. denote isolated features.

In the syntax of the language there are two sets of operators. On the one hand there are *main operators*, such as $\cdot \vee \cdot, \cdot \wedge \cdot, \mathtt{A};\cdot, \overline{\mathtt{A}};\cdot, \mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \cdot, \mathtt{A} \not\Rightarrow \mathtt{B}\ \mathtt{in}\ \cdot$, that directly correspond to relationships in FODA Diagrams. On the other hand, we have *auxiliary operators*, such as $\mathtt{nil}, \swarrow, \cdot \backslash \mathtt{A}, \cdot \Rightarrow \mathtt{A}$, which we need to define the semantics of the language.

Definition 1. A *software product line* is a term generated by the following Extended BNF-like expression:

$$\begin{aligned} P ::= \quad & \swarrow \mid \mathtt{nil} \mid \mathtt{A};P \mid \overline{\mathtt{A}};P \mid \\ & P \vee Q \mid P \wedge Q \mid \mathtt{A} \not\Rightarrow \mathtt{B}\ \mathtt{in}\ P \mid \\ & \mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P \mid P \setminus \mathtt{A} \mid P \Rightarrow \mathtt{A} \end{aligned}$$

where $\mathtt{A}, \mathtt{B} \in \mathcal{F}$. We denote the set of terms of this algebra by SPLA.

In order to avoid writing too many parentheses in the terms, we are going to assume left-associativity in binary operators and the

following precedence in the operators (from higher to lower priority): $\mathtt{A};P, \overline{\mathtt{A}};P, P \vee Q, P \wedge Q, \mathtt{A} \not\Rightarrow \mathtt{B}\ \mathtt{in}\ P, \mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P, P \setminus \mathtt{A}$, and $P \Rightarrow \mathtt{A}$. We show (Proposition 2) that the binary operators are commutative and associative. As a result, the *choose-one* operator ($\cdot \vee \cdot$) and the *conjunction* operator ($\cdot \wedge \cdot$) are *n*-ary operators instead of just binary operators.

The Extended BNF in Definition 1 says that a term of SPLA is a sequence of operators and features. An SPLA term represents sets of *products*. We introduce the formal definition of products of an SPL expressed in SPLA in Section 4. Basically, a product is a set of features that can be derived from an SPLA term. Next we explain the meaning of each operator by using some examples.

There are two terminal symbols in the language, nil and ✓, we need them to define the semantics of the language. Let us note that the products of a term in SPLA will be computed following some rules. The computation will finish when no further steps are allowed. This fact is represented by the nil symbol. We will introduce rules to compute a product, with this computation finishing when no further steps are required, a situation represented by nil During the computation of an SPLA term, we have to represent the situation in which a *valid product* of the term has been computed. This fact is represented by the ✓ symbol.

The operators $\mathtt{A};P$ and $\overline{\mathtt{A}};P$ add the feature A to any product that can be obtained from *P*. The operator $\mathtt{A};P$ indicates that A is mandatory while $\overline{\mathtt{A}};P$ indicates that A is optional. There are two binary operators: $P \vee Q$ and $P \wedge Q$. The first one represents the *choose-one* operator while the second one represents the *conjunction* operator.

Example 1. The term $\mathtt{A};\overline{\mathtt{B}};✓$ represents an SPL with two valid products. We have a product with only the feature A and another product with the features A and B. This is because feature B is optional in this case.

The term $\mathtt{A};✓ \vee (\mathtt{B};✓ \vee \mathtt{C};✓)$ has three valid products with one feature in each. The first has feature A, the second has feature B and the third has feature C.

We will show that $\vee$ is commutative and associative so we could rewrite the previous term without parentheses: $\mathtt{A};✓ \vee \mathtt{B};✓ \vee \mathtt{C};✓$. Therefore, this operator can be seen as choosing 1 feature from *n* options.

The term $\mathtt{A};(\mathtt{B};✓ \wedge \mathtt{C};✓)$ represents a mandatory relationship; and we will see that this term has only one product with three features: A, B, and C. As well as the *choose-one* operator, we will show that the $\wedge$ operator is commutative and associative. So we can consider this an *n*-ary operator.

The constraints are easily represented in SPLA. The operator $\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in} P$ represents the *require* constraint in FODA. The operator $\mathtt{A} \not\Rightarrow \mathtt{B}\ \mathtt{in}\ P$ represents the *exclusion* constraint in FODA.

Example 2. The term $\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{A};✓$ has only one valid product with the features A and B.

Let us consider $P = \mathtt{A};(\mathtt{B};✓ \vee \mathtt{C};✓)$. This term has two valid products: The first one has the features A and B, while the second one has the features A and C.

If we add to the previous term the following constraint $\mathtt{A} \not\Rightarrow \mathtt{B}\mathtt{in}P$, then this new term has only one product with the features A and C.

The operator $P \Rightarrow \mathtt{A}$ is necessary to define the behavior of the $\mathtt{A} \Rightarrow \mathtt{B}\mathtt{in}P$ operator: When we compute the products of the term $\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P$, we have to take into account whether product A has been produced or not. In the case it has been produced, we have to annotate that we need to produce B in the future. The operator $P \Rightarrow \mathtt{B}$ is used for this purpose. The same happens with the operator

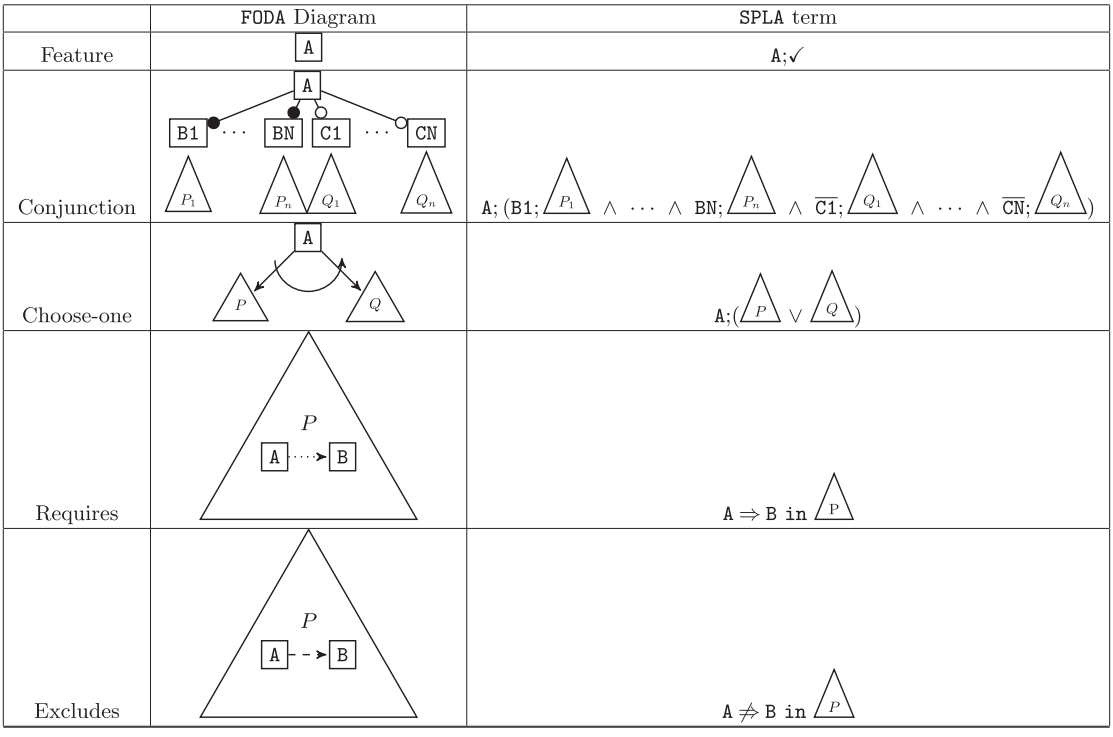

Fig. 3. Mapping from FODA diagram to SPLA.

$P\backslash$B. When we compute the products of A $\nRightarrow$ B in P, if the feature A is computed at some point, we annotate that B must not be included. The operator $P\backslash$B indicates that product B is forbidden.

3.2. Translation: from FODA to SPLA

The matching table used to translate FODA diagrams to SPLA syntax is presented in Fig. 3. Any FODA Diagram can be translated into SPLA by using the rules in this figure. The translation from FODA into SPLA is made in three steps:

1. First we make the translation of the diagram without taking into consideration any kind of restrictions.
2. Next we codify the *require* restrictions. The order in which these relations appear may be relevant. In order to overcome this problem, we introduce all the restrictions that are in the transitive closure of the original diagram: If the *require* constraints A ⋯⋯➢ B and B ⋯⋯➢ C are in the diagram, we also introduce the *require* constraint A ⋯⋯➢ C. Proposition 9 proves that, if the set of constraints are closed under transitivity then the order in which they are chosen is not important. Let us remark that this closure can be computed in $\mathcal{O}(n^3)$ with the Floyd–Warshall algorithm, with n being the number of involved features.
3. Finally we codify the *exclude* constraints. In this case Proposition 4 proves that the order in which the *exclude* constraints are chosen is not important.

This translation is correct because of Propositions 9 and 4. These propositions cannot be included here because they need the equivalence relation that will be introduced in Section 4 (Definition 6). Furthermore, their proof is easier after the results about the denotational semantics in Section 5.

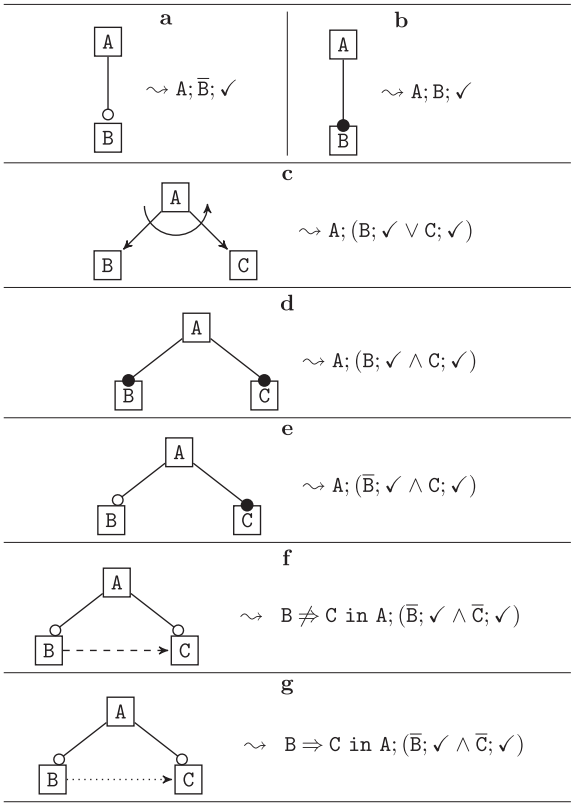

Fig. 4. Examples of translation from FODA diagrams into SPLA grammar.

In order to add clarity, Fig. 3 only considers choose-one diagrams with two choices. However, we can represent also n-ary *choose-one* diagrams because, as we have already said, the $\vee$ operator in SPLA is commutative and associative (Proposition 2).

Example 3. The translation of the FODA Diagram in Fig. 2 by using the mapping rules of Fig. 3 is presented in Fig. 4.

4. Operational semantics

So far, we only have a syntax to express the SPLs in SPLA, but for a formal study we need a semantics. In this section we define a labeled transition system for any term $P \in$ SPLA. The transitions are *annotated* with the set $\mathcal{F} \cup \{\checkmark\}$, with $\mathcal{F}$ being the set of features and $\checkmark \notin \mathcal{F}$. In particular, if A $\in \mathcal{F}$, the transition $P \xrightarrow{\text{A}} Q$ means that there is a product of P that contains the feature A. The transitions of the form $P \xrightarrow{\checkmark}$ nil mean that a product has been produced. The formal operational semantic rules of the algebra are presented in Fig. 5.

Definition 2. Let $P, Q \in$ SPLA and $a \in \mathcal{F} \cup \{\checkmark\}$. There is a transition from P to Q labeled with the symbol a, denoted by $P \xrightarrow{a} Q$, if it can be deduced from the rules in Fig. 5.

Before giving any properties of this semantics let us justify the rules in Fig. 5. We will define the products of an SPL from the set of traces obtained from the defined transitions.

First we have the rule [**tick**]. The intuitive meaning of this rule is that we have reached a point where a product of the SPL has been computed. Let us note that nil has no transitions, this means that nil does not have any valid products.

Rules [**feat**], [**ofeat**1], and [**ofeat**2] deal directly with the computation of features. Rule [**ofeat**2] means that we have a valid product without considering an optional feature, in other words, this rule is the one that establishes the difference between an optional and a mandatory feature. Feature A is optional in P because $P \xrightarrow{\text{A}} P_1$ and $P \xrightarrow{\checkmark}$ nil. In this sense, the transition $P \xrightarrow{\checkmark}$ nil indicates not only that P has already computed a valid product, but also it indicates that if P can compute any other features, these additional features are optional.

Rules [**cho**1] and [**cho**2] deal with the *choose-one* operator. These rules indicate that the computation of $P \vee Q$ must choose between the features in P or the features in Q.

Rules [**con**1], [**con**2], and [**con**3] deal with the *conjunction* operator. The main rules are [**con**1] and [**con**2]. These rules are symmetrical to each other. They indicate that any product of $P \wedge Q$ must have the features of P and Q. Rule [**con**3] indicates that both members have to agree in order to *deliver* a product.

Rules [**req1**], [**req2**], and [**req3**] deal with the *require* constraint. Rule [**req1**] indicate that A $\Rightarrow$ B in P behaves like P as long as feature A has not been computed. Rule [**req2**] indicates that B is mandatory once A has been computed. Finally [**req3**] is necessary for Lemma 1.

Rules [**excl1**] to [**excl4**] deal with the exclusion constraint. Rule [**excl1**] indicates that A $\nRightarrow$ B in P behaves like P as long as P does not compute feature A or B. Rule [**excl2**] indicates that once P produces A, feature B must be forbidden. Rule [**excl3**] indicates just the opposite: when feature B is computed, then A must be forbidden. This rule might be surprising, but there is no reason to forbid A $\nRightarrow$ B in P to compute feature B. So if A $\nRightarrow$ B in P computes feature B then feature A must be forbidden. Otherwise the exclusion constraint would not have been fulfilled.

Rules [**forb1**] and [**forb2**] deal with the auxiliary operator $P\backslash$A that forbids the computation of feature A. Let us note that there is no rule that computes A. This means that if feature A is computed by P, the computation is blocked and no products can be produced.

$$[\textbf{tick}]\ \checkmark \xrightarrow{\checkmark} \texttt{nil} \qquad [\textbf{feat}]\ \texttt{A}; P \xrightarrow{\texttt{A}} P$$

$$[\textbf{ofeat1}]\ \overline{\texttt{A}}; P \xrightarrow{\texttt{A}} P \qquad [\textbf{ofeat2}]\ \overline{\texttt{A}}; P \xrightarrow{\checkmark} \texttt{nil}$$

$$[\textbf{cho1}]\ \frac{P \xrightarrow{a} P_1}{P \vee Q \xrightarrow{a} P_1} \qquad [\textbf{cho2}]\ \frac{P \xrightarrow{a} P_1}{Q \vee P \xrightarrow{a} P_1}$$

$$[\textbf{con1}]\ \frac{P \xrightarrow{\texttt{A}} P_1}{P \wedge Q \xrightarrow{\texttt{A}} P_1 \wedge Q} \qquad [\textbf{con2}]\ \frac{P \xrightarrow{\texttt{A}} P_1}{Q \wedge P \xrightarrow{\texttt{A}} Q \wedge P_1}$$

$$[\textbf{con3}]\ \frac{P \xrightarrow{\checkmark} \texttt{nil}, Q \xrightarrow{\checkmark} \texttt{nil}}{P \wedge Q \xrightarrow{\checkmark} \texttt{nil}} \qquad [\textbf{req1}]\ \frac{P \xrightarrow{\texttt{C}} P_1,\ \texttt{C} \neq \texttt{A}}{\texttt{A} \Rightarrow \texttt{B in } P \xrightarrow{\texttt{C}} \texttt{A} \Rightarrow \texttt{B in } P_1}$$

$$[\textbf{req2}]\ \frac{P \xrightarrow{\texttt{A}} P_1}{\texttt{A} \Rightarrow \texttt{B in } P \xrightarrow{\texttt{A}} P_1 \Rightarrow \texttt{B}} \qquad [\textbf{req3}]\ \frac{P \xrightarrow{\checkmark} \texttt{nil}}{\texttt{A} \Rightarrow \texttt{B in } P \xrightarrow{\checkmark} \texttt{nil}}$$

$$[\textbf{excl1}]\ \frac{P \xrightarrow{\texttt{C}} P_1,\ \texttt{C} \neq \texttt{A} \wedge \texttt{C} \neq \texttt{B}}{\texttt{A} \not\Rightarrow \texttt{B in } P \xrightarrow{\texttt{C}} \texttt{A} \not\Rightarrow \texttt{B in } P_1} \qquad [\textbf{excl2}]\ \frac{P \xrightarrow{\texttt{A}} P_1}{\texttt{A} \not\Rightarrow \texttt{B in } P \xrightarrow{\texttt{A}} P_1 \backslash \texttt{B}}$$

$$[\textbf{excl3}]\ \frac{P \xrightarrow{\texttt{B}} P_1}{\texttt{A} \not\Rightarrow \texttt{B in } P \xrightarrow{\texttt{B}} P_1 \backslash \texttt{A}} \qquad [\textbf{excl4}]\ \frac{P \xrightarrow{\checkmark} \texttt{nil}}{\texttt{A} \not\Rightarrow \texttt{B in } P \xrightarrow{\checkmark} \texttt{nil}}$$

$$[\textbf{forb1}]\ \frac{P \xrightarrow{\texttt{B}} P_1,\ \texttt{B} \neq \texttt{A}}{P \backslash \texttt{A} \xrightarrow{\texttt{B}} P_1 \backslash \texttt{A}} \qquad [\textbf{forb2}]\ \frac{P \xrightarrow{\checkmark} \texttt{nil}}{P \backslash \texttt{A} \xrightarrow{\checkmark} \texttt{nil}}$$

$$[\textbf{mand1}]\ \frac{P \xrightarrow{\checkmark} \texttt{nil}}{P \Rightarrow \texttt{A} \xrightarrow{A} \checkmark} \qquad [\textbf{mand2}]\ \frac{P \xrightarrow{\texttt{A}} P_1}{P \Rightarrow \texttt{A} \xrightarrow{\texttt{A}} P_1}$$

$$[\textbf{mand3}]\ \frac{P \xrightarrow{\texttt{B}} P_1,\ \texttt{A} \neq \texttt{B}}{P \Rightarrow \texttt{A} \xrightarrow{\texttt{B}} P_1 \Rightarrow \texttt{A}}$$

$$\texttt{A}, \texttt{B}, \texttt{C} \in \mathcal{F},\ a \in \mathcal{F} \cup \{\checkmark\}$$

Fig. 5. Rules defining the operational semantics of SPLA.

Rules [**mand1**], [**mand2**], and [**mand3**] deal with the auxiliary operator $P \Rightarrow \texttt{A}$. Rule [**mand1**] indicates that feature A must be computed before *delivering* a product. Rules [**mand2**] and [**mand3**] indicates that $P \Rightarrow \texttt{A}$ behaves like P. We need two rules in this case because when feature A is computed it is no longer necessary to continue considering this feature as mandatory and so the operator can be removed from the term.

We can see the operational semantics of a term as a *computational* tree (see the examples in Figs. 6–8). The root is the term itself and the branches are labeled with features. The branches of the tree represent the products of the term. We obtain a valid product when we reach a node that has an outgoing arc labeled with ✓. At this point no more features can be obtained in the corresponding branch. This is what the following lemma establishes.

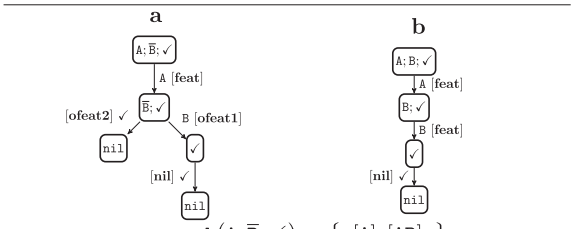

Fig. 6. Application of the operational semantic rules 1/3.

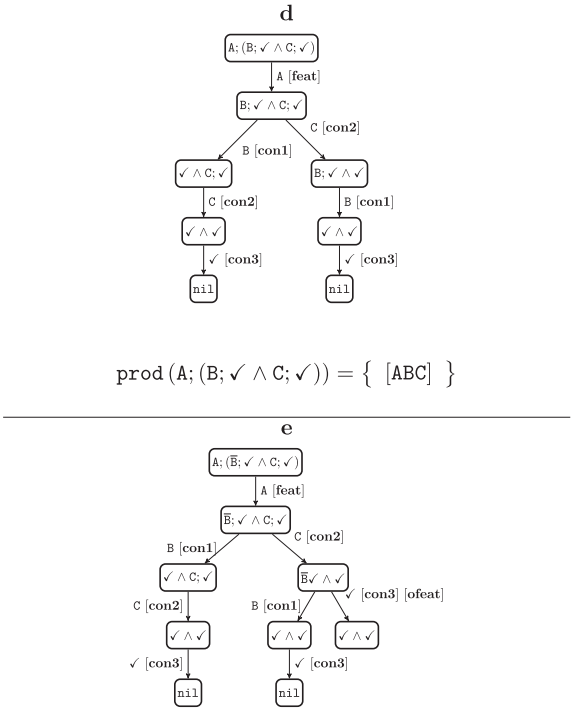

Fig. 7. Application of the operational semantic rules 2/3.

Lemma 1. *Let $P, Q \in$ SPLA, if $P \xrightarrow{\checkmark} Q$ then $Q =$* nil.

Once we have defined the operational semantics of the algebra, we can define the traces of an SPL, and from these traces we obtain its products.

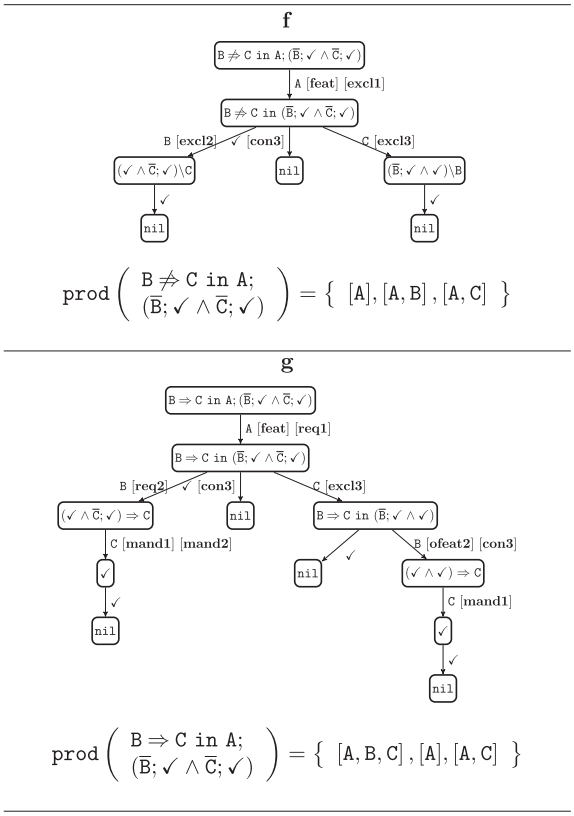

Fig. 8. Application of the operational semantic rules 3/3.

Definition 3. A trace is a sequence $s \in \mathcal{F}^*$. The empty trace is denoted by ϵ. Let s_1 and s_2 be traces, we denote the concatenation of s_1 and s_2 by $s_1 \cdot s_2$. Let $A \in \mathcal{F}$ and let s be a trace, we say that A *is in the trace* s, written $A \in s$, iff there exist traces s_1 and s_2 such that $s = s_1 \cdot A \cdot s_2$.

We can extend the transitions in Definition 2 to traces. Let P, Q, $R \in \mathtt{SPLA}$, we inductively define the transitions $P \xrightarrow{s} R$ as follows:

- $P \xrightarrow{\epsilon} P$.
- If $P \xrightarrow{A} Q$ and $Q \xrightarrow{s} R$, then $P \xrightarrow{A \cdot s} R$.

Only traces ending with the symbol $\checkmark$ can be considered as products. Therefore, in order to obtain the products of an SPL, we need to take its *successful* traces: the traces that end with the $\checkmark$ symbol. Let us note that the $\checkmark$ symbol is not a feature, so we do not include it in the trace.

Definition 4. Let $P \in \mathtt{SPLA}$ and $s \in \mathcal{F}^*$, s is a *successful trace* of P, written $s \in \mathtt{tr}(P)$, iff $P \xrightarrow{s} Q \xrightarrow{\checkmark} \mathtt{nil}$.

The order in which the features are produced cannot be represented in FODA. For this reason, different traces can define the same product. For instance, the product obtained from the trace AB is the same as the one represented by the trace BA. Thus in order to get the products of an SPL we have to consider sets that result from traces.

Definition 5. Let s be a trace. The *set induced by the trace*, written $[s]$, is the set obtained from the elements of the trace without considering their position in the trace.

Let $P \in \mathtt{SPLA}$, we define *the products of* P, written $\mathtt{prod}(P)$, as $\mathtt{prod}(P) = \{[s] | s \in \mathtt{tr}(P)\}$.

Example 4. Now let us consider the SPLA term $P = A; (\overline{B}; \checkmark \wedge \overline{C}; \checkmark)$. The possible computations of P are:

$P \xrightarrow{A} \overline{B}; \checkmark \wedge \overline{C}; \checkmark \xrightarrow{\checkmark} \mathtt{nil}$

$P \xrightarrow{A} \overline{B}; \checkmark \wedge \overline{C}; \checkmark \xrightarrow{B} \checkmark \wedge \overline{C}; \checkmark \xrightarrow{\checkmark} \mathtt{nil}$

$P \xrightarrow{A} \overline{B}; \checkmark \wedge \overline{C}; \checkmark \xrightarrow{B} \checkmark \wedge \overline{C}; \checkmark \xrightarrow{C} \checkmark \wedge \checkmark \xrightarrow{\checkmark} \mathtt{nil}$

$P \xrightarrow{A} \overline{B}; \checkmark \wedge \overline{C}; \checkmark \xrightarrow{C} \overline{B}; \checkmark \wedge \checkmark \xrightarrow{\checkmark} \mathtt{nil}$

$P \xrightarrow{A} \overline{B}; \checkmark \wedge \overline{C}; \checkmark \xrightarrow{C} \overline{B}; \checkmark \wedge \checkmark \xrightarrow{B} \checkmark \wedge \checkmark \xrightarrow{\checkmark} \mathtt{nil}$

Then $\mathtt{tr}(P) = \{A, AB, ABC, AC, ACB\}$. Therefore $\mathtt{prod}(P) = \{[A], [AB], [ABC], [AC]\}$ since $[ABC] = [ACB]$

In order to illustrate the operational semantics, we will review the examples presented in Fig. 4, giving their semantics.

Example 5. The semantics of the terms in Fig. 4 have been split in Figs. 6–8.

First let us discuss the differences between examples **a** and **b**. In example **a** feature B is optional while in **b** it is mandatory. This is reflected in example **b** by the fact that there is a branch corresponding to the transition $\overline{B}; \checkmark \xrightarrow{\checkmark} \mathtt{nil}$. This branch is not in Example **a**.

Now let us focus on Examples **c** and **d**. They show the difference between the *conjunction* operator and the *choose-one* operator. In the *choose-one* operator the member that is not needed for the computation *disappears*, while in the *conjunction* operator the other member remains. As a result Example **c** has the traces AB and AC, giving two different products [AB] and [AC]. In contrast, the traces in example **d** are ABC and ACB, giving just the product [ABC].

Once we have defined the products of an SPL, it is the time to define an equivalence relation based on products.

Definition 6. Let P, $Q \in \mathtt{SPLA}$. We say that P and Q are equivalent, written $P \equiv Q$ if the products derived from both SPLs are the same: $\mathtt{prod}(P) = \mathtt{prod}(Q)$.

Since the relation $\equiv$ is based on set equality it is also an equivalence relation. In Section 5 we will see that it is also a congruence.

Proposition 1. *Let P, Q, R* $\in$ SPLA. *The following properties hold:*

- $P \equiv P$.
- *If* $P \equiv Q$ *then* $Q \equiv P$.
- *If* $P \equiv Q$ *and* $Q \equiv R$ *then* $P \equiv R$.

Next we present some basic properties of the algebra, such as the commutativity and associativity of the binary operators. These properties are important since they allow us to extend the binary operators to n-ary operators.

Proposition 2. *Let P, Q, R* $\in$ SPLA. *The following properties hold:*

Commutativity $P \vee Q \equiv Q \vee P$ *and* $P \wedge Q \equiv Q \wedge P$.
Associativity $P \vee (Q \vee R) \equiv (P \vee Q) \vee R$ *and* $P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R$.

5. Denotational semantics

The *denotational* semantics is more abstract than the operational one, since it does not rely on computation steps. In this section we provide a denotational semantics for SPLA. In order to define this denotational semantics, the first thing to do is to establish the *mathematical domain* where the syntactical objects of SPLA will be represented.

As we have noted in Section 3, the semantics of any SPLA expression is given by its set of products, and each product can be characterized by its features. So the mathematical domain we

need is $\mathcal{P}(\mathcal{P}(\mathcal{F}))$,[1] remembering that $\mathcal{F}$ is the set of features. The next step is to define a semantic operator for any of the syntactical operators in SPLA. This is done in the following definition.

Definition 7. Let $P, Q \in \mathcal{P}(\mathcal{P}(\mathcal{F}))$ be two sets of products and let $\mathtt{A}, \mathtt{B} \in \mathcal{F}$ be two features. We define the following operators:

- $[\![\mathtt{nil}]\!] = \emptyset$
- $[\![\checkmark]\!] = \{\emptyset\}$
- $[\![\mathtt{A};\cdot]\!] : \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ as
$$[\![\mathtt{A};\cdot]\!](P) = \{\{\mathtt{A}\} \cup p | p \in P\}$$
- $[\![\overline{\mathtt{A}};\cdot]\!] : \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ as
$$[\![\overline{\mathtt{A}};\cdot]\!](P) = \{\emptyset\} \cup \{\{\mathtt{A}\} \cup p | p \in P\}$$
- $[\![\cdot \vee \cdot]\!] : \mathcal{P}(\mathcal{P}(\mathcal{F})) \times \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ as
$$[\![\cdot \vee \cdot]\!](P, Q) = P \cup Q$$
- $[\![\cdot \wedge \cdot]\!] : \mathcal{P}(\mathcal{P}(\mathcal{F})) \times \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ as
$$[\![\cdot \wedge \cdot]\!](P, Q) = \{p \cup q | p \in P,\ q \in Q\}$$
- $[\![\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \cdot]\!] : \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ as
$$[\![\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \cdot]\!](P) = \begin{array}{l}\{p | p \in P, \mathtt{A} \notin p\} \cup \\ \{p \cup \{\mathtt{B}\} | p \in P, \mathtt{A} \in p\}\end{array}$$
- $[\![\mathtt{A} \not\Rightarrow \mathtt{B}\ \mathtt{in}\ \cdot]\!] : \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ as
$$[\![\mathtt{A} \not\Rightarrow \mathtt{B}\ \mathtt{in}\ \cdot]\!](P) = \begin{array}{l}\{p | p \in P, \mathtt{A} \notin p\} \cup \\ \{p | p \in P, \mathtt{B} \notin p\}\end{array}$$
- $[\![\cdot \Rightarrow \mathtt{A}]\!] : \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ as
$$[\![\cdot \Rightarrow \mathtt{A}]\!](P) = \{p \cup \{\mathtt{A}\} | p \in P\}$$
- $[\![\cdot \backslash \mathtt{A}]\!] : \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ as
$$[\![\cdot \backslash \mathtt{A}]\!](P) = \{p | p \in P, \mathtt{A} \notin p\}$$

With these operators over sets of products, we can define the denotational semantics of any SPLA expression, which is defined inductively in the usual way.

Definition 8. The denotational semantics of SPLA is the function $[\![\cdot]\!] : \mathtt{SPLA} \to \mathcal{P}(\mathcal{P}(\mathcal{F}))$ inductively defined as follows: for any n-ary operator $\mathtt{op} \in \{\mathtt{nil}, \checkmark, \mathtt{A};\cdot, \overline{\mathtt{A}};\cdot, \cdot \vee \cdot, \cdot \wedge \cdot, \mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \cdot, \mathtt{A} \not\Rightarrow \mathtt{B}\ \mathtt{in}\ \cdot, \cdot \Rightarrow \mathtt{A}, \cdot \backslash \mathtt{A}\}$[2]:
$$[\![\mathtt{op}(P_1, \ldots P_n)]\!] = [\![\mathtt{op}]\!]([\![P_1]\!], \ldots, [\![P_n]\!])$$

Example 6. In order to illustrate the denotational semantics, we apply it to the examples presented in Fig. 4. The results are presented in Fig. 9.

The rest of this section is devoted to proving that the set of products computed by the operational semantics coincides with the one computed by the denotational semantics. In order to do it, first we need some auxiliary results that relate the operational semantics with the denotational operators from Definition 7.

The first result deals with the termination of a trace. The products of an SPL are computed from the bottom up. That means that the first product *computed* by the denotational semantics is the product with no features. Let us note that $\{\emptyset\} = \mathtt{prod}(\checkmark) = [\![\checkmark]\!]$, but $\emptyset = \mathtt{prod}(\mathtt{nil}) = [\![\mathtt{nil}]\!]$.

Lemma 2. *Let us consider* $P \in \mathtt{SPLA}$, *if* $P \xrightarrow{\checkmark} \mathtt{nil}$ *then* $\emptyset \in [\![P]\!]$.

[1] If X is a set, $\mathcal{P}(X)$ denotes the power set of X.

[2] nil and $\checkmark$ are 0-ary operators; $\mathtt{A};\cdot, \overline{\mathtt{A}};\cdot \mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}\,\cdot, \mathtt{A} \not\Rightarrow \mathtt{B}\,\mathtt{in}\,\cdot, \cdot \Rightarrow \mathtt{A}, \cdot \backslash \mathtt{A}$ are 1-ary operators; $\cdot \vee \cdot$ and $\cdot \wedge \cdot$ are 2-ary operators.

a

$[\![\checkmark]\!] = \{\emptyset\}$
$[\![\overline{\mathtt{B}};\checkmark]\!] = [\![\overline{\mathtt{B}};\cdot]\!]([\![\checkmark]\!]) = [\![\overline{\mathtt{B}};\cdot]\!](\{\emptyset\}) = \{\emptyset, \{\mathtt{B}\}\}$
$[\![\mathtt{A};\overline{\mathtt{B}};\checkmark]\!] = [\![\mathtt{A};\cdot]\!]([\![\overline{\mathtt{B}};\checkmark]\!]) = [\![\mathtt{A};\cdot]\!](\{\emptyset\} \cup \{\mathtt{B}\}) = \{\{\mathtt{A}\}, \{\mathtt{A}, \mathtt{B}\}\}$

b

$[\![\mathtt{B};\checkmark]\!] = [\![\mathtt{B};\cdot]\!]([\![\checkmark]\!]) = [\![\mathtt{B};\cdot]\!](\{\emptyset\}) = \{\{\mathtt{B}\}\}$
$[\![\mathtt{A};\mathtt{B};\checkmark]\!] = [\![\mathtt{A};\cdot]\!]([\![\mathtt{B};\checkmark]\!]) = [\![\mathtt{A};\cdot]\!](\{\mathtt{B}\}) = \{\{\mathtt{A}, \mathtt{B}\}\}$

c

$[\![\mathtt{B};\checkmark]\!] = \{\{\mathtt{B}\}\}$
$[\![\mathtt{C};\checkmark]\!] = \{\{\mathtt{C}\}\}$
$[\![\mathtt{B};\checkmark \vee \mathtt{C};\checkmark]\!] = [\![\cdot \vee \cdot]\!]([\![\mathtt{B};\checkmark]\!], [\![\mathtt{C};\checkmark]\!]) = [\![\mathtt{B};\checkmark]\!] \cup [\![\mathtt{C};\checkmark]\!] = \{\{\mathtt{B}\}, \{\mathtt{C}\}\}$
$[\![\mathtt{A};(\mathtt{B};\checkmark \vee \mathtt{C};\checkmark)]\!] = [\![\mathtt{A};\cdot]\!]([\![\mathtt{B};\checkmark \vee \mathtt{C};\checkmark]\!]) = [\![\mathtt{A};\cdot]\!](\{\{\mathtt{B}\}, \{\mathtt{C}\}\}) = \{\{\mathtt{A}, \mathtt{B}\}, \{\mathtt{A}, \mathtt{C}\}\}$

d

$[\![\mathtt{B};\checkmark \wedge \mathtt{C};\checkmark]\!] = [\![\cdot \wedge \cdot]\!]([\![\mathtt{B};\checkmark]\!], [\![\mathtt{C};\checkmark]\!]) = [\![\cdot \wedge \cdot]\!](\{\{\mathtt{B}\}\}, \{\{\mathtt{C}\}\}) = \{\{\mathtt{B}, \mathtt{C}\}\}$
$[\![\mathtt{A};(\mathtt{B};\checkmark \wedge \mathtt{C};\checkmark)]\!] = [\![\mathtt{A};\cdot]\!](\{\{\mathtt{B}, \mathtt{C}\}\}) = \{\{\mathtt{A}, \mathtt{B}, \mathtt{C}\}\}$

e

$[\![\overline{\mathtt{B}};\checkmark \wedge \mathtt{C};\checkmark)]\!] = [\![\cdot \wedge \cdot]\!](\{\emptyset, \{\mathtt{B}\}\}, \{\{\mathtt{C}\}\}) = \{\{\mathtt{C}\}, \{\mathtt{B}, \mathtt{C}\}\}$
$[\![\mathtt{A};(\overline{\mathtt{B}};\checkmark \wedge \mathtt{C};\checkmark)]\!] = [\![\mathtt{A};\cdot]\!]([\![\overline{\mathtt{B}};\checkmark \wedge \mathtt{C};\checkmark)]\!]) = [\![\mathtt{A};\cdot]\!](\{\{\mathtt{C}\}, \{\mathtt{B}, \mathtt{C}\}\}) = \{\{\mathtt{A}, \mathtt{C}\}, \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}\}$

f

$[\![\overline{\mathtt{B}};\checkmark \wedge \overline{\mathtt{C}};\checkmark]\!] = [\![\cdot \wedge \cdot]\!]([\![\overline{\mathtt{B}};\checkmark]\!], [\![\overline{\mathtt{C}};\checkmark]\!]) = [\![\cdot \wedge \cdot]\!](\{\emptyset, \{\mathtt{B}\}\}, \{\emptyset, \{\mathtt{C}\}\}) = \{\emptyset, \{\mathtt{B}\}, \{\mathtt{C}\}, \{\mathtt{B}, \mathtt{C}\}\}$
$[\![\mathtt{A};(\overline{\mathtt{B}};\checkmark \wedge \overline{\mathtt{C}};\checkmark)]\!] = [\![\mathtt{A};\cdot]\!]([\![\overline{\mathtt{B}};\checkmark \wedge \overline{\mathtt{C}};\checkmark]\!]) = [\![\mathtt{A};\cdot]\!](\{\emptyset, \{\mathtt{B}\}, \{\mathtt{C}\}, \{\mathtt{B}, \mathtt{C}\}\}) = \{\{\mathtt{A}\}, \{\mathtt{A}, \mathtt{B}\}, \{\mathtt{A}, \mathtt{C}\}, \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}\}$
$\left[\!\!\left[\begin{array}{l}\mathtt{B} \not\Rightarrow \mathtt{C}\ \mathtt{in}\ \mathtt{A}; \\ (\overline{\mathtt{B}};\checkmark \wedge \overline{\mathtt{C}};\checkmark)\end{array} \right]\!\!\right] = [\![\mathtt{B} \not\Rightarrow \mathtt{C}\ \mathtt{in}\ \cdot]\!] \left(\begin{array}{l}\{\{\mathtt{A}\}, \{\mathtt{A}, \mathtt{B}\}, \\ \{\mathtt{A}, \mathtt{C}\}, \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}\}\end{array} \right) = \{\{\mathtt{A}\}, \{\mathtt{A}, \mathtt{C}\}, \{\mathtt{A}, \mathtt{B}\}\}$

g

$\left[\!\!\left[\begin{array}{l}\mathtt{B} \Rightarrow \mathtt{C}\ \mathtt{in}\ \mathtt{A}; \\ (\overline{\mathtt{B}};\checkmark \wedge \overline{\mathtt{C}};\checkmark)\end{array} \right]\!\!\right] = [\![\mathtt{B} \Rightarrow \mathtt{C}\ \mathtt{in}\ \cdot]\!] \left(\begin{array}{l}\{\{\mathtt{A}\}, \{\mathtt{A}, \mathtt{B}\}, \\ \{\mathtt{A}, \mathtt{C}\}, \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}\}\end{array} \right) = \{\{\mathtt{A}\}, \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}, \{\mathtt{A}, \mathtt{C}\}\}$

Fig. 9. Application of the denotational semantic rules.

Next we will present a lemma for each operator of the syntax. Each of these lemmas indicates that the corresponding semantic operator is well defined in Definition 7. These results will be needed in the inductive case of Theorem 1.

Lemma 3. *Let* $P, P' \in \mathtt{SPLA}$, *and* $\mathtt{A}, \mathtt{B} \in \mathcal{F}$, *then*

1. $\mathtt{prod}(\mathtt{A};P) = [\![\mathtt{A};]\!](\mathtt{prod}(P))$
2. $\mathtt{prod}(\overline{\mathtt{A}};P) = [\![\mathtt{A};]\!](\mathtt{prod}(P))$
3. $\mathtt{prod}(P \vee P') = [\![\vee]\!](\mathtt{prod}(P), \mathtt{prod}(P'))$.
4. $\mathtt{prod}(P \wedge P') = [\![\wedge]\!](\mathtt{prod}(P), \mathtt{prod}(P'))$.
5. $\mathtt{prod}(P \Rightarrow \mathtt{A}) = [\![\cdot \Rightarrow \mathtt{A}]\!](\mathtt{prod}(P))$.
6. $\mathtt{prod}(P \backslash \mathtt{A}) = [\![\cdot \backslash \mathtt{A}]\!](\mathtt{prod}(P))$.
7. $\mathtt{prod}(\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}\,P) = [\![\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}\,\cdot]\!](\mathtt{prod}(P))$.
8. $\mathtt{prod}(\mathtt{A} \not\Rightarrow \mathtt{B}\,\mathtt{in}\,P) = [\![\mathtt{A} \not\Rightarrow \mathtt{B}\,\mathtt{in}\,\cdot]\!](\mathtt{prod}(P))$.

Now we have the result we were looking for: The denotational semantics and the operational semantics are equivalent.

Theorem 1. *Let $P \in \mathtt{SPLA}$, then $\mathtt{prod}(P) = [\![P]\!]$.*

An immediate result from the previous theorem is that the equivalence relation $\equiv$ is a congruence.

Corollary 1. *The equivalence relation $\equiv$ is a congruence: For any n-ary operator $\mathtt{op}$, and $P_1, \ldots P_n$, $Q_1, \ldots Q_n \in \mathtt{SPLA}$ such that $P_1 \equiv Q_1, \ldots, P_n \equiv Q_n$, we have*

$$\mathtt{op}(P_1, \ldots, P_n) \equiv \mathtt{op}(Q_1, \ldots, Q_n)$$

5.1. Correctness of the translation

The translation procedure in Section 3.2 does not impose an order among the restrictions. This means that one diagram can be translated in different syntactical terms in SPLA. In this section we are going to prove that all that different terms are equivalent indeed. In the translation, we first codify the require constraints from the FODA diagram. When all the *require* constraints are codified, then the *exclude* constraints are codified. First, let us prove the property that indicates the order in which the *require* constraints are chosen. Let us recall that when making the translation from FODA to SPLA we consider the closure of the *require* constraints.

First, let us prove the property that indicates the order in which the *require* constraints are chosen. Let us recall that when making the translation from FODA to SPLA we are going to consider the closure of the *require* constraints.

Definition 9. Let $P \in \mathtt{SPLA}$ be term, we say that is *closed with respect the require constraints* if has the following form:

$$\mathtt{A}_1 \Rightarrow \mathtt{B}_1 \ \mathtt{in}\ \mathtt{A}_2 \Rightarrow \mathtt{B}_2 \ \mathtt{in}\ \cdots \mathtt{A}_n \Rightarrow \mathtt{B}_n \ \mathtt{in}\ Q$$

where Q has no restrictions and the set of restrictions is closed by transitivity, that is, if there are features $\mathtt{A}, \mathtt{B}, \mathtt{C} \in \mathcal{F}$ and $1 \leqslant i,j \leqslant n$ such that $\mathtt{A} = \mathtt{A}_i$, $\mathtt{B} = \mathtt{B}_i = \mathtt{A}_j$ and $\mathtt{C} = \mathtt{B}_j$ then there is $1 \leqslant k \leqslant n$ such that $\mathtt{A} = \mathtt{A}_k$ and $\mathtt{C} = \mathtt{B}_k$.

Proposition 3. *Let $P \in \mathtt{SPLA}$ be a closed term with respect to the require constraints. Let us consider $Q' \in \mathtt{SPLA}$ by reordering the require constraints in P. Under these conditions $P \equiv Q$.*

We also have to prove that the order in which the *exclude* constraints are chosen is irrelevant. This is because two *exclude* constrains in FODA are always interchangeable.

Proposition 4. *Let $P \in \mathtt{SPLA}$ be a term and $\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D} \in \mathcal{F}$, then $\mathtt{A} \nRightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{C} \nRightarrow \mathtt{D}\ \mathtt{in} P \equiv \mathtt{C} \nRightarrow \mathtt{D}\ \mathtt{in}\ \mathtt{A} \nRightarrow \mathtt{B}\ \mathtt{in} P$.*

5.2. Full abstraction

Finally in this Section, we show that the model is fully abstract: given any set S of products, there is a SPLA term whose semantics is exactly the set S. Moreover, this term can be constructed by using a subset of the grammar: $\mathtt{nil}, \checkmark$, the prefix operator, and the choice operator.

Definition 10. Let $P \in \mathtt{SPLA}$, we say that it is a *basic term* if it can be generated by the following grammar

$$P ::= \checkmark | \mathtt{nil} | \mathtt{A}; P | P \vee Q$$

We denote the set of basic terms as $\mathtt{SPLA}_b$.

The result that we are looking for is the following.

Theorem 2. *Let $\mathcal{F}$ be a finite set of features and let $A \in \mathcal{P}(\mathcal{P}(\mathcal{F}))$, there exists $P \in \mathtt{SPLA}_b$ such that $\mathtt{prod}(P) = A$.*

This result indicates also an important practical consequence. Let us imagine any other operator that could be added to the syntax of SPLA. This result indicates that this operator can be *derived* from the operators in $\mathtt{SPLA}_b$. Even the operators in SPLA can be rewritten in terms of the operators in $\mathtt{SPLA}_b$. In fact, in the next section we show how we can remove the non-basic operators from any SPLA term.

This result establishes also an interesting theoretical consequence. Since the denotational semantics is fully abstract, it is isomorphic to the initial model with respect to the equivalence relation. That is, the set of products with the operators of the denotational semantics is isomorphic to the algebra of terms: $\mathtt{SPLA}/\equiv$.

6. Axiomatic semantics

In this section we give an axiomatic semantics for SPLA, presenting *sound* and *complete* axioms for the language. As usual, *soundness* means that the equalities deduced from the axiom system are indeed correct: $P =_E Q$ implies $P \equiv Q$. The completeness means that all the identities can be deduced from the axiom system, that is $P \equiv Q$ implies $P =_E Q$.

Definition 11. Let P, $Q \in \mathtt{SPLA}$. We say that we *deduce the equivalence* of P and Q if $P =_E Q$ can be deduced from the set of equations in Figs. 10–13.

To prove the soundness it is enough to show that the operators are congruent (Theorem 1) and that each axiom is correct.

Proposition 5. *Let $\mathtt{A}, \mathtt{B} \in \mathcal{F}$ be two features, and P and Q be terms of SPLA. The equations in Figs. 10–13 are correct.*

To prove the completeness we need the concept of *normal forms*. In order to define the normal forms we prove that some operators are *derived* from the *basic* operators. These basic operators are the base ones ($\mathtt{nil}$ and $\checkmark$), the prefix operator ($\mathtt{A}; P$), and the *choose-one* operator ($P \vee Q$). The following example shows how some operators can be removed.

Example 7. Let us consider the following SPL $P = \mathtt{A}; \checkmark \wedge \mathtt{B}; \checkmark$. It is easy to compute its successful traces which are {AB,BA}, so $\mathtt{prod}(P) = \{[\mathtt{AB}]\}$. This SPL has the same products as $\mathtt{A}; \mathtt{B}; \checkmark$.

[REQ1] $\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ (\mathtt{C}; P) =_E \mathtt{C}; (\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P)$

[REQ2] $\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ (\mathtt{A}; P) =_E \mathtt{A}; (P \Rightarrow \mathtt{B})$

[REQ3] $\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ (\mathtt{B}; P) =_E \mathtt{B}; P$

[REQ4] $\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P \vee Q =_E\ (\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P) \vee (\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ Q)$

[REQ5] $\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \checkmark =_E \checkmark$

[REQ6] $\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{nil} =_E \mathtt{nil}$

[MAND1] $(\mathtt{A}; P) \Rightarrow \mathtt{A} =_E \mathtt{A}; P$

[MAND2] $(\mathtt{B}; P) \Rightarrow \mathtt{A} =_E \mathtt{B}; (P \Rightarrow \mathtt{A})$

[MAND3] $\checkmark \Rightarrow \mathtt{A} =_E \mathtt{A}; \checkmark$

[MAND4] $\mathtt{nil} \Rightarrow \mathtt{A} =_E \mathtt{nil}$

[MAND5] $(P \vee Q) \Rightarrow \mathtt{A} =_E (P \Rightarrow \mathtt{A}) \vee (Q \Rightarrow \mathtt{A})$

Fig. 10. Equations to remove *require* and *mandatory* operators.

[EXCL1] $\mathtt{A} \not\Rightarrow \mathtt{B}$ in $(\mathtt{C}; P) =_E \mathtt{C}; (\mathtt{A} \not\Rightarrow \mathtt{B}$ in $P)$

[EXCL2] $\mathtt{A} \not\Rightarrow \mathtt{B}$ in $(\mathtt{A}; P) =_E \mathtt{A}; (P \backslash \mathtt{B})$

[EXCL3] $\mathtt{A} \not\Rightarrow \mathtt{B}$ in $(\mathtt{B}; P) =_E \mathtt{B}; (P \backslash \mathtt{A})$

[EXCL4] $\mathtt{A} \not\Rightarrow \mathtt{B}$ in $P \vee Q =_E (\mathtt{A} \not\Rightarrow \mathtt{B}$ in $P) \vee (\mathtt{A} \not\Rightarrow \mathtt{B}$ in $Q)$

[EXCL5] $\mathtt{A} \not\Rightarrow \mathtt{B}$ in $\checkmark =_E \checkmark$

[EXCL6] $\mathtt{A} \not\Rightarrow \mathtt{B}$ in $\mathtt{nil} =_E \mathtt{nil}$

[FORB1] $(\mathtt{A}; P) \backslash \mathtt{A} =_E \mathtt{nil}$

[FORB2] $(\mathtt{B}; P) \backslash \mathtt{A} =_E \mathtt{B}; (P \backslash \mathtt{A})$

[FORB3] $\checkmark \backslash \mathtt{A} =_E \checkmark$

[FORB4] $\mathtt{nil} \backslash \mathtt{A} =_E \mathtt{nil}$

[FORB5] $(P \vee Q) \backslash \mathtt{A} =_E (P \backslash \mathtt{A}) \vee (Q \backslash \mathtt{A})$

Fig. 11. Equations to remove exclusion, and forbid operators.

[CON1] $(\mathtt{A}; P) \wedge Q =_E \mathtt{A}; (P \wedge Q)$

[CON2] $P \wedge Q =_E Q \wedge P.$

[CON3] $P \wedge (Q \vee R) =_E (P \wedge Q) \vee (P \wedge R).$

[CON4] $P \wedge \mathtt{nil} =_E \mathtt{nil}$

[CON5] $P \wedge \checkmark =_E P$

Fig. 12. Axioms to remove the *conjunction* operator.

[PRE1] $\mathtt{A}; \mathtt{B}; P =_E \mathtt{B}; \mathtt{A}; P.$

[PRE2] $\overline{\mathtt{A}}; P =_E (\mathtt{A}; P) \vee \checkmark.$

[PRE3] $(\mathtt{A}; P) \vee (\mathtt{A}; Q) =_E \mathtt{A}; (P \vee Q)$

[PRE4] $\mathtt{A}; \mathtt{nil} =_E \mathtt{nil}$

[PRE5] $\mathtt{A}; \mathtt{A}; P =_E \mathtt{A}; P$

[CHO1] $P \vee Q =_E Q \vee P.$

[CHO2] $(P \vee Q) \vee R =_E P \vee (Q \vee R).$

[CHO3] $P \vee \mathtt{nil} =_E P.$

[CHO4] $P \vee P = P.$

Fig. 13. Axioms for basic operators and optional features.

Indeed, by applying the indicated axioms we have the following deduction

$$\begin{aligned} &\mathtt{A}; \checkmark \wedge \mathtt{B}; \checkmark =_E && [\textbf{CON1}] \\ &\mathtt{A}; (\checkmark \wedge \mathtt{B}; \checkmark) =_E && [\textbf{CON2}] \\ &\mathtt{A}; (\mathtt{B}; \checkmark \wedge \checkmark) =_E && [\textbf{CON1}] \\ &\mathtt{A}; \mathtt{B}; (\checkmark \wedge \checkmark) =_E && [\textbf{CON5}] \\ &\mathtt{A}; \mathtt{B}; \checkmark \end{aligned}$$

The set of axioms in Figs. 10–12, plus the axiom [**PRE 2**] in Fig. 13 allow the non-basic operators (Definition 10) to be removed from any $P \in \mathtt{SPLA}$. The idea is to prove that $Q \in \mathtt{SPLA}_b$ exists such that $P \equiv Q$.

Let us suppose that we have a term $P \in \mathtt{SPLA}$ that contains a non-basic operator. Then we can find $Q \in \mathtt{SPLA}$ where either, the non-basic operator has disappeared or it is deeper in the syntactic tree of Q. Then iterating this process we can make all non-basic operators disappear. Then we have the theorem we are looking for.

Theorem 3. *Let $P \in \mathtt{SPLA}$, there exists $Q \in \mathtt{SPLA}_b$ such that $P =_E Q$.*

Since we know how to remove the non-basic operators of an SPLA term, we focus on proving the completeness restricted to basic terms. To do so, we define are our *normal forms*, and then we prove that any basic term can be transformed to a normal form by using the basic axioms in Fig. 13.

In order to give the formal definitions of normal forms we provide auxiliary definitions. First we assume that there is an order relation $\leqslant \subseteq \mathcal{F} \times \mathcal{F}$ that must be isomorphic to the natural numbers in case $\mathcal{F}$ is infinite. Next, we need the *vocabulary* of a basic SPLA term, that is the set of features appearing in the expression.

Definition 12. Let $P, Q \in \mathtt{SPLA}_b$ be two basic SPLA terms. We define the *vocabulary* as the function $\mathtt{voc} : \mathtt{SPLA}_b \to \mathcal{P}(\mathcal{F})$ defined inductively as:

- $\mathtt{voc}(\mathtt{nil}) = \mathtt{voc}(\checkmark) = \emptyset$
- $\mathtt{voc}(\mathtt{A}; P) = \{\mathtt{A}\} \cup \mathtt{voc}(P)$
- $\mathtt{voc}(P \vee Q) = \mathtt{voc}(P) \cup \mathtt{voc}(Q)$

Before giving the definition of normal forms, we define a simpler case that is the case of pre-normal forms.

Definition 13. A basic SPLA term $P \in \mathtt{SPLA}_b$ is in *pre-normal form*, written $P \in \mathtt{SPLA}_{pre}$, iff it has one of the following forms.

1. $\mathtt{nil}, \checkmark$, or
2. There exists $n > 0, \{\mathtt{A}_1, \ldots, \mathtt{A}_n\} \subseteq \mathcal{F}$, and there exist $P_1, \ldots P_n \in \mathtt{SPLA}_{pre}$ with $P_i \neq \mathtt{nil}$ for $1 \leqslant i \leqslant n$ and $\{\mathtt{A}_1, \ldots \mathtt{A}_n\} \cap \mathtt{voc}(P_j) = \emptyset$ for $1 \leqslant j \leqslant n$ and either

$$P = (\mathtt{A}_1; P_1) \vee \cdots (\mathtt{A}_n; P_n)$$

or

$$P = (\mathtt{A}_1; P_1) \vee \cdots (\mathtt{A}_n; P_n) \vee \checkmark$$

In this case we say that the features $\{\mathtt{A}_1, \ldots, \mathtt{A}_n\}$ are at the top level of P.

Next we present an auxiliary lemma that will be used in Proposition 8. This lemma establishes that if a feature appears in the vocabulary of a pre-normal form then it appears in at least one product of the pre-normal form. Let us note that this result is not true[3] in ordinary terms because of the restrictions that might appear in the terms.

Lemma 4. *Let $P \in \mathtt{SPLA}_{pre}$, then*

$$\mathtt{voc}(P) = \{\mathtt{A} | \mathtt{A} \in p,\ p \in \mathtt{prod}(P)\}$$

The next lemma establishes that if a feature A appears in a pre-normal form P, the normal form can be transformed into another equivalent normal form Q so that A is at the top level of the syntax tree of Q.

[3] The vocabulary of an ordinary term has not been formally defined. Definition 12 could easily be extended to the set of features appearing in the syntax of a term.

Lemma 5. *Let $P \in \mathtt{SPLA}_{pre}$ and let $\mathtt{A} \in \mathtt{voc}(P)$. Then there is $Q \in \mathtt{SPLA}_{pre}$ such that $P \equiv Q$ and $\mathtt{A} \in \{\mathtt{A}_1, \ldots, \mathtt{A}_n\}$ according to condition 2 of Definition 13 applied to Q.*

The next proposition establishes the first result we need to prove the completeness: Any term can be transformed into an equivalent pre-normal form. The result is restricted to basic terms but, because of Theorem 3, it can be extended to any ordinary term.

Proposition 6. *Let $P \in \mathtt{SPLA}_b$, there exists a pre-normal form $Q \in \mathtt{SPLA}_{pre}$ such that $P =_E Q$.*

The problem with pre-normal forms is that there are syntactically different expressions that are equivalent, as the following example shows.

Example 8. Let us consider the following $\mathtt{SPLA}_b$ expressions:

$P = (\mathtt{A}; \mathtt{C}; \checkmark) \vee \mathtt{B}; \checkmark$
$Q = (\mathtt{C}; \mathtt{A}; \checkmark) \vee \mathtt{B}; \checkmark$

Both expressions are in pre-normal form and both are equivalent.

The way to obtain a unique normal form for any $\mathtt{SPLA}_b$ expression is to use the above mentioned order among features. Let us assume $\mathtt{A} < \mathtt{B} < \mathtt{C}$; in this case we say that P is in normal form while Q is not.

Normal forms are a particular case of pre-normal forms that overcome this problem. They make use of the order required between features.

Definition 14. Let us consider $P \in \mathtt{SPLA}_{pre}$. We will say that P is a *normal form*, written $P \in \mathtt{SPLA}_{nf}$ iff $P = \mathtt{nil}, P = \checkmark$ or if the sets $\{\mathtt{A}_1, \ldots, \mathtt{A}_n\}$ and $\{P_1, \ldots, P_n\}$ in Definition 13.2 satisfy:

- $\mathtt{A}_i < \mathtt{A}_j$ for $1 \leqslant i < j \leqslant n$.
- $\mathtt{A}_i < \mathtt{B}$ for any $\mathtt{B} \in \mathtt{voc}(P_j)$ for $1 \leqslant i \leqslant j \leqslant n$.

Example 9. Figs. 14–16 show the normal forms corresponding to the examples in Fig. 4 assuming that $\mathtt{A} < \mathtt{B} < \mathtt{C}$. The examples **a**, **b**, and **c** are not included due to the fact that they are already normal forms.

Now we have our first result. Any expression can be transformed into a normal form.

Proposition 7. *Let $P \in \mathtt{SPLA}_{pre}$. Then there exists a normal form $Q \in \mathtt{SPLA}_{nf}$ such that $P =_E Q$.*

The next result shows that two normal forms that are semantically equivalent, are also identical at the lexical level.

d

$$\begin{array}{rll}
\mathtt{A}; (\mathtt{B}; \checkmark \wedge \mathtt{C}; \checkmark) =_E & [\mathtt{CON1}] & \\
\mathtt{A}; \mathtt{B}; (\checkmark \wedge \mathtt{C}; \checkmark) =_E & [\mathtt{CON2}], & [\mathtt{CON5}] \\
\mathtt{A}; \mathtt{B}; \mathtt{C}; \checkmark & &
\end{array}$$

e

$$\begin{array}{rll}
\mathtt{A}; (\overline{\mathtt{B}}; \checkmark \wedge \mathtt{C}; \checkmark) =_E & [\mathtt{CON2}], & [\mathtt{CON1}] \\
\mathtt{A}; \mathtt{C}; (\checkmark \wedge \overline{\mathtt{B}}; \checkmark) =_E & [\mathtt{CON2}], & [\mathtt{CON5}] \\
\mathtt{A}; \mathtt{C}; \overline{\mathtt{B}}; \checkmark =_E & [\mathtt{PRE2}] & \\
\mathtt{A}; \mathtt{C}; (\mathtt{B}; \checkmark \vee \checkmark) =_E & [\mathtt{PRE3}] & \\
\mathtt{A}; (\mathtt{C}; \mathtt{B}; \checkmark \vee \mathtt{C}; \checkmark) =_E & [\mathtt{PRE1}] & \\
\mathtt{A}; (\mathtt{B}; \mathtt{C}; \checkmark \vee \mathtt{C}; \checkmark) &
\end{array}$$

Fig. 14. Transformation to normal form 1/3.

f

$$\begin{array}{rll}
\mathtt{A}; (\overline{\mathtt{B}}; \checkmark \wedge \overline{\mathtt{C}}; \checkmark) =_E & [\mathtt{PRE2}] & \\
\mathtt{A}; ((\mathtt{B}; \checkmark \vee \checkmark) \wedge (\mathtt{C}; \checkmark \vee \checkmark)) =_E & [\mathtt{CON3}] & \\
\mathtt{A}; ((\mathtt{B}; \checkmark \wedge \mathtt{C}; \checkmark) \vee & & \\
(\mathtt{B}; \checkmark \wedge \checkmark) \vee (\checkmark \wedge \mathtt{C}; \checkmark) \vee (\checkmark \wedge \checkmark)) =_E & [\mathtt{CON1}] & \\
 & [\mathtt{CON5}] & \\
\mathtt{A}; ((\mathtt{B}; \mathtt{C}; \checkmark) \vee (\mathtt{B}; \checkmark) \vee (\mathtt{C}; \checkmark) \vee \checkmark) &
\end{array} \quad (1)$$

$$\begin{array}{rll}
\mathtt{B} \not\Rightarrow \mathtt{C} \text{ in } \mathtt{A}; (\overline{\mathtt{B}}; \checkmark \wedge \overline{\mathtt{C}}; \checkmark) =_E & (1) & \\
\mathtt{A}; \mathtt{B} \not\Rightarrow \mathtt{C} \text{ in } \begin{pmatrix} (\mathtt{B}; \mathtt{C}; \checkmark) \vee \\ (\mathtt{B}; \checkmark) \vee (\mathtt{C}; \checkmark) \vee \checkmark \end{pmatrix} =_E & [\mathtt{EXCL4}] & \\
\mathtt{A}; \begin{pmatrix} (\mathtt{B} \not\Rightarrow \mathtt{C} \text{ in } \mathtt{B}; \mathtt{C}; \checkmark) \vee \\ (\mathtt{B} \not\Rightarrow \mathtt{C} \text{ in } \mathtt{B}; \checkmark) \vee \\ (\mathtt{B} \not\Rightarrow \mathtt{C} \text{ in } \mathtt{C}; \checkmark) \vee \\ (\mathtt{B} \not\Rightarrow \mathtt{C} \text{ in } \checkmark) \end{pmatrix} =_E & \begin{matrix}[\mathtt{EXCL1}] \\ [\mathtt{EXCL2}] \\ [\mathtt{EXCL5}]\end{matrix} & \\
\mathtt{A}; \begin{pmatrix} (\mathtt{B}; (\mathtt{C}; \checkmark \backslash \mathtt{C})) \vee \\ (\mathtt{B}; (\checkmark \backslash \mathtt{C})) \vee (\mathtt{C}; (\checkmark \backslash \mathtt{B})) \vee \checkmark \end{pmatrix} =_E & \begin{matrix}[\mathtt{FORB1}] \\ [\mathtt{FORB3}]\end{matrix} & \\
\mathtt{A}; ((\mathtt{B}; \mathtt{nil}) \vee (\mathtt{B}; \checkmark) \vee (\mathtt{C}; \checkmark) \vee \checkmark) =_E & [\mathtt{PRE4}] & \\
\mathtt{A}; (\mathtt{nil} \vee (\mathtt{B}; \checkmark) \vee (\mathtt{C}; \checkmark) \vee \checkmark) =_E & [\mathtt{CHO3}] & \\
\mathtt{A}; ((\mathtt{B}; \checkmark) \vee (\mathtt{C}; \checkmark) \vee \checkmark) &
\end{array}$$

Fig. 15. Transformation to normal form 2/3.

g

$$\begin{array}{rll}
\mathtt{B} \Rightarrow \mathtt{C} \text{ in } \mathtt{A}; (\overline{\mathtt{B}}; \checkmark \wedge \overline{\mathtt{C}}; \checkmark) =_E & (1) & \\
\mathtt{A}; \mathtt{B} \Rightarrow \mathtt{C} \text{ in } \begin{pmatrix} (\mathtt{B}; \mathtt{C}; \checkmark) \vee \\ (\mathtt{B}; \checkmark) \vee (\mathtt{C}; \checkmark) \vee \checkmark \end{pmatrix} =_E & [\mathtt{EXCL4}] & \\
\mathtt{A}; \begin{pmatrix} (\mathtt{B} \Rightarrow \mathtt{C} \text{ in } \mathtt{B}; \mathtt{C}; \checkmark) \vee \\ (\mathtt{B} \Rightarrow \mathtt{C} \text{ in } \mathtt{B}; \checkmark) \vee \\ (\mathtt{B} \Rightarrow \mathtt{C} \text{ in } \mathtt{C}; \checkmark) \vee \\ (\mathtt{B} \Rightarrow \mathtt{C} \text{ in } \checkmark) \end{pmatrix} =_E & \begin{matrix}[\mathtt{REQ1}] \\ [\mathtt{REQ2}] \\ [\mathtt{REQ5}]\end{matrix} & \\
\mathtt{A}; \begin{pmatrix} (\mathtt{B}; (\mathtt{C}; \checkmark \Rightarrow \mathtt{C})) \vee \\ (\mathtt{B}; (\checkmark \Rightarrow \mathtt{C})) \vee (\mathtt{C}; \checkmark) \vee \checkmark \end{pmatrix} =_E & \begin{matrix}[\mathtt{MAND1}] \\ [\mathtt{MAND3}]\end{matrix} & \\
\mathtt{A}; \begin{pmatrix} (\mathtt{B}; \mathtt{C}; \checkmark) \vee \\ (\mathtt{B}; \mathtt{C}; \checkmark) \vee (\mathtt{C}; \checkmark) \vee \checkmark \end{pmatrix} =_E & [\mathtt{CHO4}] & \\
\mathtt{A}; (\mathtt{B}; \mathtt{C}; \checkmark \vee \mathtt{C}; \checkmark \vee \checkmark) &
\end{array}$$

Fig. 16. Transformation to normal form 3/3.

Proposition 8. *Let $P, Q \in \mathtt{SPLA}_{nf}$. If they are semantically equivalent, $P \equiv Q$, then they are syntactically identical $P = Q$.*

Finally, we can prove the main result of this section: The deductive system is sound and complete.

Theorem 4. *Let us consider $P,Q \in \mathtt{SPLA}$. Then $P \equiv Q$ if and only if $P =_E Q$.*

7. Checking satisfiability

In this section we present a mechanism to check the *satisfiability* of a syntactical term, that is, whether there is a product satisfying all restrictions imposed by the term.

Definition 15. Let $P \in \mathtt{SPLA}$. We say that P is *satisfiable* iff $\mathtt{prod}(P) \neq \emptyset$.

Checking the satisfiability of any $P \in \mathtt{SPLA}$ could be done by computing the products P, by using the rules defining the

denotational semantics. Once we have this set we could check whether it is empty or not. However, computing all products may be not feasible. So, in this section we present an alternative that uses a SAT-solver. From any $P \in \mathtt{SPLA}$ we build a propositional formula $\phi(P)$ such that P is satisfiable if and only if there exists a valuation v such that $v \models \phi(P)$. In building such a formula we keep track of the order in which features are produced. Any feature $\mathtt{A}$ is associated with a set of boolean variables: $\mathtt{A}_k$ for $k \in \mathbb{N}$. The integer associated with the feature is used to keep track of the order in which it has been produced. Therefore, our boolean variables will have the form $\mathtt{A}_k$ where $\mathtt{A} \in \mathcal{F}$ and $k \in \mathbb{N}$.

Before describing how compute the formula associated to a syntactical term, we need some auxiliary definitions. The maxin function is especially important because it is used to compute the next index available for a feature in a given formula.

Definition 16. Let φ be a propositional formula, we denote *the set of boolean variables appearing in* φ by $\mathsf{vars}(\varphi)$.

Let $\mathtt{A} \in \mathcal{F}$ and φ be a propositional formula. We define the function that returns the *maximum index of* $\mathtt{A}$ *in the formula* φ as follows:

$$\mathsf{maxin}(\mathtt{A}, \varphi) = \begin{cases} k & \exists l \in \mathbb{N} : \mathtt{A}_l \in \mathsf{vars}(\varphi), \\ & k = \max\{l | \mathtt{A}_l \in \mathsf{vars}(\varphi)\} \\ -1 & \text{otherwise} \end{cases}$$

Finally, if $l < 0$, A_l will denote the symbol $\bot$.

Lemma 7 is necessary in order to prove the main result of this section. But in order to prove Lemma 7 we need to *complete* the computed formulas in the presence of the choice operator. It is convenient that the function maxin is same in both members of a choice operator, this is achieved by the completing of a formula.

Definition 17. Given φ_1 and φ_2, we define the *completion of* φ_1 *up to* φ_2, written $\varphi_1^{\Rrightarrow \varphi_2}$, as follows:

$$\varphi_1 \wedge \bigwedge_{\substack{\mathtt{A} \in \mathcal{F}, \\ l = \mathsf{maxin}(\mathtt{A}, \varphi_2), \\ k = \mathsf{maxin}(\mathtt{A}, \varphi_1), \\ 0 \leqslant k < l}} (\neg \mathtt{A}_l \rightarrow \neg \mathtt{A}_{l-1}) \wedge \cdots \wedge (\neg \mathtt{A}_{k+1} \rightarrow \neg \mathtt{A}_k)$$

The first consequence of the previous definition is that $\varphi_1^{\Rrightarrow \varphi_2}$ is stronger than φ_1: if $v \models \varphi_1^{\Rrightarrow \varphi_2}$ then $v \models \varphi_1$. let us note that in the previous definition, the new variables do not belong to φ_1. So any valuation v such that $v \models \varphi_1$ can be extended to a new valuation v' in such a way that it only modifies the value of the new variables and $v' \models \varphi_1^{\Rrightarrow \varphi_2}$. It is also important to note that the number of variables does not increase due to this completion. This is because the variables that we introduce in $\varphi_1^{\Rrightarrow \varphi_2}$ are already in φ_2. These properties are expressed in the following lemma.

Lemma 6. *Let* φ_1 *and* φ_2 *be two propositional formulas.*

1. Let v be a valuation such that $v \models \varphi_1^{\Rrightarrow \varphi_2}$, then $v \models \varphi_1$.
2. Let v be a valuation such that $v \models \varphi_1$. Then there is a valuation v' such that $v' \models \varphi_1^{\Rrightarrow \varphi_2}$ and $v'(\mathtt{A}_l) = v(\mathtt{A}_l)$ for any feature $\mathtt{A}$ and $0 \leqslant l \leqslant maxin(\mathtt{A}, \varphi_1)$.
3. Let $\mathtt{A}$ be a feature such that there is $k \in \mathbb{N}$ satisfying $\mathtt{A}_k \in \mathsf{vars}(\varphi_1^{\Rrightarrow \varphi_2})$ but $\mathtt{A}_k \notin \mathsf{vars}(\varphi_1)$, then $\mathtt{A}_k \in \mathsf{vars}(\varphi_2)$. So $\mathsf{maxin}(\mathtt{A}, \varphi_1^{\Rrightarrow \varphi_2}) = \mathsf{maxin}(\mathtt{A}, \varphi_2)$

Definition 18. Let $P \in \mathtt{SPLA}$, we define its associated propositional formula, written $\phi(P)$, as follows:

$$\phi(\mathtt{nil}) = \bot$$
$$\phi(\blacktriangleright) = \top$$

$$\phi(\mathtt{A}; P) = \mathtt{A}_{l+1} \wedge \phi(P) \text{ where } l = \max(0, \mathsf{maxin}(\mathtt{A}, \phi(P)))$$

$$\phi(\overline{\mathtt{A}}; P) = \top$$
$$\phi(P \vee Q) = \phi(P)^{\Rrightarrow \phi(Q)} \vee \phi(Q)^{\Rrightarrow \phi(P)}$$
$$\phi(P \wedge Q) = \phi(P) \wedge \phi(Q)$$

$$\phi(\mathtt{A} \Rightarrow \mathtt{B} \text{ in } P) = (\neg \mathtt{A}_{l+1} \rightarrow \neg \mathtt{A}_l) \wedge (\neg \mathtt{B}_{m+1} \rightarrow \neg \mathtt{B}_m) \wedge (\mathtt{A}_{l+1} \rightarrow \mathtt{B}_{m+1}) \wedge \phi(P)$$
$$\text{where } l = \mathsf{maxin}(\mathtt{A}, \phi(P)) \text{ and } m = \mathsf{maxin}(\mathtt{B}, \phi(P))$$

$$\phi(\mathtt{A} \not\Rightarrow \mathtt{B} \text{ in } P) = (\neg \mathtt{A}_{l+1} \rightarrow \neg \mathtt{A}_l) \wedge (\neg \mathtt{B}_{m+1} \rightarrow \neg \mathtt{B}_m) \wedge (\neg \mathtt{A}_{l+1} \vee \neg \mathtt{B}_{m+1}) \wedge \phi(P)$$
$$\text{where } l = \mathsf{maxin}(\mathtt{A}, \phi(P)) \text{ and } m = \mathsf{maxin}(\mathtt{B}, \phi(P))$$

$$\phi(P \Rightarrow \mathtt{A}) = \mathtt{A}_{l+1} \wedge \phi(P) \text{ where } l = \max(0, \mathsf{maxin}(\mathtt{A}, \phi(P)))$$
$$\phi(P \setminus \mathtt{A}) = \neg \mathtt{A}_l \wedge \phi(P) \textit{ that} \text{where } l = \mathsf{maxin}(\mathtt{A}, \phi(P))$$

Before giving the result that will help us to check the satisfiability of any $P \in \mathtt{SPLA}$, we need a preliminary property of $\phi(P)$.

Lemma 7. *Let* $P \in \mathtt{SPLA}, \mathtt{A} \in \mathcal{F}, v$ *be a valuation such that* $v \models \phi(P)$ *and* $l \in \mathbb{N}$ *such that* $v(\mathtt{A}_l) = 0$ *and* $\mathtt{A}_l \in \mathsf{vars}(\phi(P))$, *then* $v(\mathtt{A}_k) = 0$ *for* $k \leqslant l$.

We want to prove that there exists $p \in \mathtt{SPLA}$ if and only if the formula $\phi(P)$ is satisfiable. But there are problems in the presence of restrictions (requires, excludes, mandatory or forbid) inside a conjunction operator. So the result is restricted to syntactical terms that do not have restrictions inside the conjunction operator. This is not a major drawback since the restrictions can be considered to be external to the other operators.

Definition 19. Let $P \in \mathtt{SPLA}$, we say that is *a safe* $\mathtt{SPL}$ if there are no restrictions inside a conjunction operator ($\wedge$).

The main result of this section is Theorem 5. This Theorem relates the satisfiability of an SPL P and the satisfiability of its associated formula $\phi(P)$. This theorem is a direct consequence of Propositions 9 and 10. Proposition 9 is the *left to right implication* of Theorem 5. It is proven by structural induction on P, and an extra condition is needed to prove the result.

Proposition 9. *Let* $P \in \mathtt{SPLA}$ *be a safe* $\mathtt{SPL}$. *If* $p \in \mathtt{prod}(P)$ *then there is a valuation* v *such that* $v \models \phi(P)$, *and* $\mathtt{A} \in p$ *iff* $k \geqslant 0$ *and* $v(\mathtt{A}_k) = 1$ *where* $k = \mathsf{maxin}(\mathtt{A}, \phi(P))$

Proposition 10 is the *right to left implication* of Theorem 5. It is also proven by structural induction on P. As in the previous case, an extra condition is needed to prove the result.

Proposition 10. *Let* $P \in \mathtt{SPLA}$ *be a safe* $\mathtt{SPL}$. *If there is a valuation* v *such that* $v \models \phi(P)$ *then there is a product* $p \in \mathtt{prod}(P)$ *such that* $\mathtt{A} \notin p$ *for any feature* $\mathtt{A}$ *satisfying* $v(\mathtt{A}_l) = 0$ *for all* $0 \leqslant l \mathsf{maxin}(\mathtt{A}, \phi(P))$.

So, finally we have the result we require.

Theorem 5. *Let* $P \in \mathtt{SPLA}$ *be a safe* $\mathtt{SPL}$, *then* $p \in \mathtt{prod}(P)$ *iff there is a valuation* v *such that* $v \models \phi(P)$.

Finally, in Section 9 we present the results of a tool implementing this result. This tool uses a SAT-solver. Let us note that $\phi(P)$ is not a CNF formula. So, in order to check its satisfiability with a SAT-solver, $\phi(P)$ has to be converted into CNF.

8. Study of VSS system

In this section we consider the example of a model of a Video Streaming Software. The FODA Diagram of our Video Streaming Software is presented in Fig. 17. This system incorporates the following features: VSS, TBR, VCC, 720Kbps, 256Kbps, H.264 and MPEG.2. The *initial feature*[4] for this SPL is VSS (Video Streaming Software). Any product of this SPL will need this first feature. Let us note that each feature has a unique name the definition of which appears in the domain terminology dictionary of *Video Streaming Software*.

Next we present the features, and we indicate their intuitive meanings and relationships between them. The initial first feature is VSS. Its related features are: TBR (the Transmission Bit-Rate) and VCC (the Video Codec) features. There is a *mandatory* relationship between them (it is represented by using an arc). This relationship establishes that this set of features is included in all implementations of the software where the VSS is included. There are additional *mandatory* relationships in the diagram. For instance, the feature 720Kbps has a relationship with its *parent*. There are other features in this system that are included as optional. For instance, the feature 256Kbps is optional, meaning that this feature may be (or may not be) included in the final products of the SPL. Let us note that the inclusion of this feature depends on other relationships that represent constraints. Finally, let us consider the features H.264 and MPEG.2. The *choose-one* operators relates to them. This means that at least one of this features will be presented in the final product of the SPL. In this system, there is a set of *rules* that denotes some *restrictions*. For instance, when the feature MPEG.2 is included, the feature 720Kbps is also included. Furthermore, when the feature H.264 is selected, the feature 256Kbps must not appear. Let us consider the constraint that relates MPEG.2 with the speed 720Kbps. It is a *require* constraint, that is, if the MPEG.2 feature is selected, then the speed of the TBR must be 720Kbps. Let us note that this constraint is not necessary since 720Kbps is mandatory, we have included it to show that sometimes FODA allows the inclusion of repetitive (or useless) information. There is another constraint that focuses on the features H.264 and 256Kbps and it is an *exclusion* constraint meaning that if the feature H.264 appears then the final product cannot contain 256Kbps. Finally, a *valid* product of this model is a set of features that fulfills all the constraints in the diagram. For instance, let us consider the following products: $\mathtt{pr}_1$ consists of VSS, TBR, 720Kbps, VCC, and H.264, $\mathtt{pr}_2$ consists of VSS, TBR, 720Kbps, VCC, and MPEG.2 and $\mathtt{pr}_3$ consists of VSS, TBR, 720Kbps, 256Kbps, VCC, and H.264. In this case, $\mathtt{pr}_1$ and $\mathtt{pr}_2$ are *valid* products of this model, but $\mathtt{pr}_3$ is not a valid product of this model.

In order to present the formal analysis, first we focus on the *translation process*. Formally, the complete translation of this model into our algebra is P_1:

$$P_1 := \mathtt{MPEG.2} \Rightarrow \mathtt{720Kbps\ in}\ \mathtt{H.264} \not\Rightarrow \mathtt{256Kbps\ in}\ P_2$$

where

$$P_2 := \mathtt{VSS};(P_{21} \wedge P_{22})$$
$$P_{21} := \mathtt{TBR};(\mathtt{720Kbps};\checkmark \wedge \overline{\mathtt{256Kbps}};\checkmark)$$
$$P_{22} := \mathtt{VCC};(\mathtt{H.264}\ ;\checkmark \vee \mathtt{MPEG.2}\ ;\checkmark)$$

We depict the labeled transition system associated with the term P_1 in Fig. 18. In this figure we have skipped the subtrees (1) and (2) because we did not obtain new products from those subtrees. From the tree depicted we obtain the following traces:

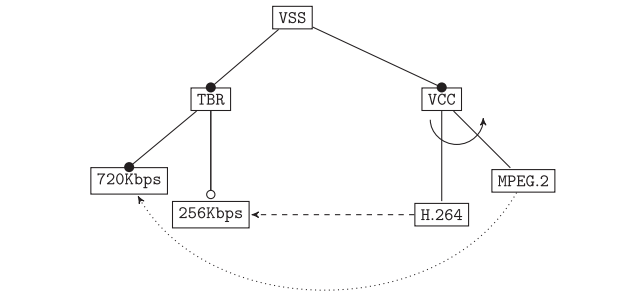

Fig. 17. Video Streaming Software – FODA representation.

- VSS TBR VCC 720Kbps H.264,
- VSS TBR VCC 720Kbps MPEG.2 720Kbps,
- VSS TBR VCC 720Kbps MPEG.2 256Kbps 720Kbps,
- VSS TBR VCC MPEG.2 720Kbps, and
- VSS TBR VCC MPEG.2 720Kbps 256Kbps.

From those traces we obtain the products:

- {VSS, TBR, VCC, 720Kbps, H.264},
- {VSS, TBR, VCC, 720Kbps, MPEG.2}, and
- {VSS, TBR, VCC, 720Kbps, MPEG.2, 256Kbps}

The denotational semantics for P_1 is presented in Fig. 19. Let us note that the set of products is not modified after applying the last semantic operator $[\![\mathtt{MPEG.2} \Rightarrow \mathtt{720Kbps\ in}\ \cdot\,]\!]$. This means that the *require* restriction is not necessary in this case. As expected (see Theorem 1), the set of products obtained by applying the denotational semantics coincides with the set of products computed with the operational semantics: $\mathtt{prod}(P_1) = [\![P_1]\!]$.

Finally, we present the *deduction process* induced by the axioms in Fig. 20. We can observe how the initial expression P_1 is transformed until we obtain a pre-normal form (Fig. 20, Eq. (5)):

$$\mathtt{VSS};\mathtt{TBR};\mathtt{VCC};\mathtt{720Kbps};(\mathtt{H.264}\ ;\checkmark \vee \mathtt{MPEG.2}\ ;\mathtt{720Kbps};(\mathtt{256Kbps};\checkmark \vee \checkmark))$$

This term is not in normal form for two reasons. First we have not established an order among the features, instead we can assume the following ordering: VSS < TBR < VCC < 720Kbps < H.264 < MPEG.2 < 256Kbps. Secondly, the repetition of feature 720Kbps in this term is not allowed in normal forms. We obtain the normal from by applying the rules in Eq. (6), and the resulting term is:

$$\mathtt{VSS};\mathtt{TBR};\mathtt{VCC};\mathtt{720Kbps};(\mathtt{H.264}\ ;\checkmark \vee \mathtt{MPEG.2}\ ;(\mathtt{256Kbps};\checkmark \vee \checkmark))$$

This is a normal form for the above ordering.

9. The SPLA Tool

In this section we present the SPLA Tool, called AT. The core of this tool is implemented in JAVA. The tool can be downloaded from http://simba.fdi.ucm.es/at and its license is GPL v3.[5]

The tool has modules that will be presented later. In order to validate them, we generate SPLs using the JAVA BeTTy Feature Model Generator[6] [40]. In BeTTy, the models are generated by setting several input parameters. We have used the following values: The percentage of constraints was 30%, the probability of a feature

[4] Sometime called the hard-system of the SPL.

[5] More details in http://www.gnu.org/copyleft/gpl.html.

[6] Can be downloaded from http://www.isa.us.es/betty/betty-online.

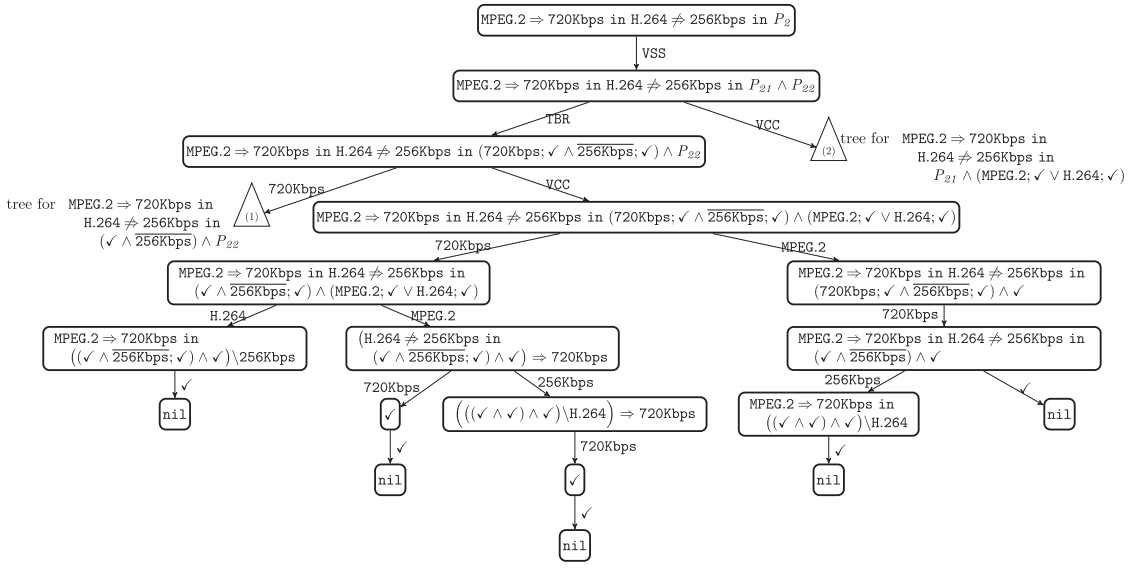

Fig. 18. Labeled transitions system of the video streaming software.

$$
\begin{array}{rcl}
[\![\checkmark]\!] & = & \{\varnothing\} \\
[\![\texttt{H.264}; \checkmark]\!] & = & \{\{\texttt{H.264}\} \cup \varnothing\} \;=\; \{\{\texttt{H.264}\}\} \\
[\![\texttt{MPEG.2}; \checkmark]\!] & = & \{\{\texttt{MPEG.2}\}\} \\
[\![\texttt{720Kbps}; \checkmark]\!] & = & \{\{\texttt{720Kbps}\}\} \\
[\![\overline{\texttt{256Kbps}}; \checkmark]\!] & = & \{\varnothing\} \cup \{\{\texttt{256Kbps}\} \cup \varnothing\} \;=\; \{\varnothing, \{\texttt{256Kbps}\}\}
\end{array}
$$

$$
\begin{array}{rcl}
[\![P_{22}]\!] & = & [\![\texttt{VCC}; (\texttt{H.264}; \checkmark \vee \texttt{MPEG.2}; \checkmark)]\!] \;=\; [\![\texttt{VCC}; \cdot]\!]\big([\![\vee]\!](\{\{\texttt{H.264}\}\}, \{\{\texttt{MPEG.2}\}\})\big) = \\
 & & [\![\texttt{VCC}; \cdot]\!]\big([\![\texttt{H.264}; \checkmark]\!] \cup [\![\texttt{MPEG.2}; \checkmark]\!]\big) \;=\; [\![\texttt{VCC}; \cdot]\!]\big(\{\{\texttt{H.264}\}\}, \{\{\texttt{MPEG.2}\}\}\big) = \\
 & & \{\{\texttt{VCC}, \texttt{H.264}\}, \{\texttt{VCC}, \texttt{MPEG.2}\}\} \\
[\![P_{21}]\!] & = & [\![\texttt{TBR}; (\texttt{720Kbps}; \checkmark \wedge \overline{\texttt{256Kbps}}; \checkmark)]\!] \;= \\
 & & [\![\texttt{TBR}; \cdot]\!]\big([\![\wedge]\!](\{\{\texttt{720Kbps}\}, \{\varnothing, \{\texttt{256Kbps}\}\})\big) \;= \\
 & & [\![\texttt{TBR}; \cdot]\!]\big(\{\{\texttt{720Kbps}\}, \{\texttt{720Kbps}, \texttt{256Kbps}\}\}\big) = \\
 & & \{\{\texttt{TBR}, \texttt{720Kbps}\}, \{\texttt{TBR}, \texttt{720Kbps}, \texttt{256Kbps}\}\}
\end{array}
$$

$$
\begin{array}{rcl}
[\![P_{21} \wedge P_{22}]\!] & = & [\![\wedge]\!]\left(\begin{array}{l}\{\{\texttt{VCC}, \texttt{H.264}\}, \{\texttt{VCC}, \texttt{MPEG.2}\}\}, \{\{\texttt{TBR}, \texttt{720Kbps}\}, \\ \{\texttt{TBR}, \texttt{720Kbps}, \texttt{256Kbps}\}\}\end{array}\right) = \\
 & & \left\{\begin{array}{l}\{\texttt{VCC}, \texttt{TBR}, \texttt{H.264}, \texttt{720Kbps}\}, \{\texttt{VCC}, \texttt{TBR}, \texttt{H.264}, \texttt{720Kbps}, \texttt{256Kbps}\}, \\ \{\texttt{VCC}, \texttt{TBR}, \texttt{MPEG.2}, \texttt{720Kbps}\}, \{\texttt{VCC}, \texttt{TBR}, \texttt{MPEG.2720Kbps}, \texttt{256Kbps}\}\end{array}\right\} \\
[\![P_2]\!] & = & [\![\texttt{VSS}; (P_{21} \wedge P_{22})]\!] = [\![\texttt{VSS};]\!](P_{21} \wedge P_{22}) = \\
 & & \left\{\begin{array}{l}\{\texttt{VSS}, \texttt{VCC}, \texttt{TBR}, \texttt{H.264}, \texttt{720Kbps}\}, \{\texttt{VSS}, \texttt{VCC}, \texttt{TBR}, \texttt{H.264}, \texttt{720Kbps}, \texttt{256Kbps}\}, \\ \{\texttt{VSS}, \texttt{VCC}, \texttt{TBR}, \texttt{MPEG.2}, \texttt{720Kbps}\}, \{\texttt{VSS}, \texttt{VCC}, \texttt{TBR}, \texttt{MPEG.2720Kbps}, \texttt{256Kbps}\}\end{array}\right\}
\end{array}
$$

$$
\begin{array}{rcl}
[\![\texttt{H.264} \not\Rightarrow \texttt{256Kbps in } P_2]\!] & = & [\![\texttt{H.264} \not\Rightarrow \texttt{256Kbps in } \cdot]\!]([\![P_2]\!]) = \\
 & & \left\{\begin{array}{l}\{\texttt{VSS}, \texttt{VCC}, \texttt{TBR}, \texttt{H.264}, \texttt{720Kbps}\}, \\ \{\texttt{VSS}, \texttt{VCC}, \texttt{TBR}, \texttt{MPEG.2}, \texttt{720Kbps}\}, \\ \{\texttt{VSS}, \texttt{VCC}, \texttt{TBR}, \texttt{MPEG.2}, \texttt{720Kbps}, \texttt{256Kbps}\}\end{array}\right\} \\
[\!\!\left[\begin{array}{l}\texttt{MPEG.2} \Rightarrow \texttt{720Kbps in} \\ \quad \texttt{H.264} \not\Rightarrow \texttt{256Kbps in } P_2\end{array}\right]\!\!] & = & [\![\texttt{MPEG.2} \Rightarrow \texttt{720Kbps in } \cdot]\!]([\![\texttt{H.264} \not\Rightarrow \texttt{256Kbps in } P_2]\!]) = \\
 & & \left\{\begin{array}{l}\{\texttt{VSS}, \texttt{VCC}, \texttt{TBR}, \texttt{H.264}, \texttt{720Kbps}\}, \{\texttt{VSS}, \texttt{VCC}, \texttt{TBR}, \texttt{MPEG.2}, \texttt{720Kbps}\}, \\ \{\texttt{VSS}, \texttt{VCC}, \texttt{TBR}, \texttt{MPEG.2}, \texttt{720Kbps}, \texttt{256Kbps}\}\end{array}\right\}
\end{array}
$$

Fig. 19. Denotational semantics of video streaming software.

being mandatory was 25%, the probability of a feature being in a *choose-one* relation was 50%.

9.1. Satisfiability module

This module computes the satisfiability of an SPL by using the results in Section 9.1. So it computes the formula associated with an SPL. Then, by using a SAT-solver, we check the satisfiability of the formula. Since the module has been developed in JAVA, we have used the SAT4j.

The experiments were run with a number of different features ranging from 1000 to 13,500. Fig. 21 shows the time needed to compute the satisfiability of the SPLs with respect to the number of features in the model.

9.2. Denotational semantic module

This module computes the products of an SPLA term according to the semantics presented in this paper.

The experiments were run with a number of different features ranging from 50 to 300. The results are in Fig. 22. There are three columns of values in this table: The first one contains the number of features, the second one contains the time required to complete the experiment, and the last one contain the number of products needed to give an answer. This number is a lower bound of the total number of features of the model with the property that if it is 0 then the model is not satisfiable. A dash in the last column indicates that we did not obtain any answer after a 15 min timeout.

$$
\begin{aligned}
&\texttt{TBR};(\texttt{720Kbps};\checkmark \wedge \overline{\texttt{256Kbps}};\checkmark)=_E && [\textbf{CON1}]\\
&\texttt{TBR};\texttt{720Kbps};(\checkmark \wedge \overline{\texttt{256Kbps}};\checkmark)=_E && [\textbf{CON5}]\ [\textbf{CON2}]\\
&\texttt{TBR};\texttt{720Kbps};\overline{\texttt{256Kbps}};\checkmark=_E && [\textbf{PRE2}]\\
&\texttt{TBR};\texttt{720Kbps};(\texttt{256Kbps};\checkmark \vee \checkmark)
\end{aligned}
\tag{2}
$$

$$
\begin{aligned}
P_2 = \texttt{VSS};(P_{21} \wedge P_{22})=_E && (2)\\
\texttt{VSS};\big(\texttt{TBR};\texttt{720Kbps};(\texttt{256Kbps};\checkmark \vee \checkmark)\big) \wedge \big(\texttt{VCC};(\texttt{H.264};\checkmark \vee \texttt{MPEG.2};\checkmark)\big)=_E && [\textbf{CON1}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big((\texttt{256Kbps};\checkmark \vee \checkmark) \wedge (\texttt{H.264};\checkmark \vee \texttt{MPEG.2};\checkmark)\big)=_E && [\textbf{CON3}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(((\texttt{256Kbps};\checkmark \vee \checkmark) \wedge \texttt{H.264};\checkmark) \vee ((\texttt{256Kbps};\checkmark \vee \checkmark) \wedge \texttt{MPEG.2};\checkmark)=_E && [\textbf{CON1}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\Big(\texttt{H.264};\big((\texttt{256Kbps};\checkmark \vee \checkmark) \wedge \checkmark\big) \vee \texttt{MPEG.2};\big((\texttt{256Kbps};\checkmark \vee \checkmark) \wedge \checkmark\big)\Big)=_E && [\textbf{CON5}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};(\texttt{256Kbps};\checkmark \vee \checkmark) \vee \texttt{MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark)\big)
\end{aligned}
\tag{3}
$$

$$
\begin{aligned}
&\texttt{H.264} \not\Rightarrow \texttt{256Kbps in } P_2=_E && (3)\\
&\texttt{H.264} \not\Rightarrow \texttt{256Kbps in } \Big(\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};(\texttt{256Kbps};\checkmark \vee \checkmark) \vee \texttt{MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark)\big)\Big)=_E && [\textbf{EXCL1}]\\
&\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\texttt{H.264} \not\Rightarrow \texttt{256Kbps in } \Big(\big(\texttt{H.264};(\texttt{256Kbps};\checkmark \vee \checkmark) \vee \texttt{MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark)\big)\Big)=_E && [\textbf{EXCL4}]\\
&\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big((\texttt{H.264} \not\Rightarrow \texttt{256Kbps in H.264};(\texttt{256Kbps};\checkmark \vee \checkmark)) \vee\\
&\qquad \vee(\texttt{H.264} \not\Rightarrow \texttt{256Kbps in MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark))\big)=_E && [\textbf{EXCL2}]\\
&\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big((\texttt{H.264};((\texttt{256Kbps};\checkmark \vee \checkmark)\backslash\texttt{256Kbps})) \vee\\
&\qquad (\texttt{H.264} \not\Rightarrow \texttt{256Kbps in MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark))\big)=_E && [\textbf{EXCL1}]\\
&\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big((\texttt{H.264};((\texttt{256Kbps};\checkmark \vee \checkmark)\backslash\texttt{256Kbps})) \vee\\
&\qquad (\texttt{MPEG.2};\texttt{H.264} \not\Rightarrow \texttt{256Kbps in } (\texttt{256Kbps};\checkmark \vee \checkmark))\big)=_E && [\textbf{FORB1}]\ [\textbf{FORB3}]\ [\textbf{FORB5}]\\
&\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};\checkmark \vee (\texttt{MPEG.2};\texttt{H.264} \not\Rightarrow \texttt{256Kbps in } (\texttt{256Kbps};\checkmark \vee \checkmark))\big)=_E && [\textbf{EXCL3}]\\
&\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big((\texttt{H.264};((\texttt{256Kbps};\checkmark \vee \checkmark)\backslash\texttt{256Kbps})) \vee\\
&\qquad (\texttt{MPEG.2};\texttt{H.264} \not\Rightarrow \texttt{256Kbps in } (\texttt{256Kbps};\checkmark \vee \checkmark))\big)=_E && [\textbf{FORB1}]\ [\textbf{FORB3}]\ [\textbf{FORB5}]\\
&\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};\checkmark \vee (\texttt{MPEG.2};\texttt{H.264} \not\Rightarrow \texttt{256Kbps in } (\texttt{256Kbps};\checkmark \vee \checkmark))\big)=_E && [\textbf{EXCL3}]\ [\textbf{EXCL4}]\ [\textbf{EXCL5}]\\
&\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};\checkmark \vee (\texttt{MPEG.2};((\texttt{256Kbps};\checkmark)\backslash\texttt{H.264} \vee \checkmark))\big)=_E && [\textbf{FORB2}]\ [\textbf{FORB3}]\ [\textbf{FORB5}]\\
&\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};\checkmark \vee (\texttt{MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark))\big)
\end{aligned}
\tag{4}
$$

$$
\begin{aligned}
P_1 = \texttt{MPEG.2} \Rightarrow \texttt{720Kbps in H.264} \not\Rightarrow \texttt{256Kbps in } P_2=_E && (4)\\
\texttt{MPEG.2} \Rightarrow \texttt{720Kbps in VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};(\texttt{H.264};\checkmark \vee \texttt{MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark)) && [\textbf{REQ1}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\texttt{MPEG.2} \Rightarrow \texttt{720Kbps in } (\texttt{H.264};\checkmark \vee \texttt{MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark)) && [\textbf{REQ4}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big((\texttt{MPEG.2} \Rightarrow \texttt{720Kbps in H.264};\checkmark) \vee (\texttt{MPEG.2} \Rightarrow \texttt{720Kbps in MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark))\big) && [\textbf{REQ1}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};(\texttt{MPEG.2} \Rightarrow \texttt{720Kbps in } \checkmark) \vee (\texttt{MPEG.2} \Rightarrow \texttt{720Kbps in } (\texttt{256Kbps};\checkmark \vee \checkmark))\big) && [\textbf{REQ5}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};\checkmark \vee (\texttt{MPEG.2} \Rightarrow \texttt{720Kbps in MPEG.2};)\big) && [\textbf{REQ2}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};\checkmark \vee \texttt{MPEG.2};((\texttt{256Kbps};\checkmark \vee \checkmark) \Rightarrow \texttt{720Kbps})\big) && [\textbf{MAND5}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};\checkmark \vee \texttt{MPEG.2};((\texttt{256Kbps};\checkmark \Rightarrow \texttt{720Kbps}) \vee (\checkmark \Rightarrow \texttt{720Kbps}))\big) && [\textbf{MAND5}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};\checkmark \vee \texttt{MPEG.2};((\texttt{256Kbps};\checkmark \Rightarrow \texttt{720Kbps}) \vee (\checkmark \Rightarrow \texttt{720Kbps}))\big) && [\textbf{MAND2}]\ [\textbf{MAND3}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};(\texttt{H.264};\checkmark \vee \texttt{MPEG.2};\texttt{720Kbps};(\texttt{256Kbps};\checkmark \vee \checkmark))=_E && [\textbf{PRE2}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};\checkmark \vee \texttt{MPEG.2};\texttt{720Kbps};(\texttt{256Kbps};\checkmark \vee \checkmark)\big)
\end{aligned}
\tag{5}
$$

$$
\begin{aligned}
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};\checkmark \vee \texttt{MPEG.2};\texttt{720Kbps};(\texttt{256Kbps};\checkmark \vee \checkmark)\big)=_E && [\textbf{PRE1}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};\checkmark \vee \texttt{720Kbps};\texttt{MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark)\big)=_E && [\textbf{PRE3}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\big(\texttt{720Kbps};\texttt{H.264};\checkmark \vee \texttt{720Kbps};\texttt{720Kbps};\texttt{MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark)\big)=_E && [\textbf{PRE5}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\big(\texttt{720Kbps};\texttt{H.264};\checkmark \vee \texttt{720Kbps};\texttt{MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark)\big)=_E && [\textbf{PRE5}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\big(\texttt{720Kbps};\texttt{H.264};\checkmark \vee \texttt{720Kbps};\texttt{MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark)\big)=_E && [\textbf{PRE5}]\\
\texttt{VSS};\texttt{TBR};\texttt{VCC};\texttt{720Kbps};\big(\texttt{H.264};\checkmark \vee \texttt{MPEG.2};(\texttt{256Kbps};\checkmark \vee \checkmark)\big)
\end{aligned}
\tag{6}
$$

Fig. 20. Deduction rules applied to the video streaming software.

Features	Time (ms.)	Features	Time (ms.)
01000	67	07500	529
01500	71	08000	556
02000	105	08500	682
02500	140	09000	428
03000	164	09500	639
03500	153	10000	620
04000	193	10500	396
04500	410	11000	513
05000	331	11500	575
05500	339	12000	552
06000	441	12500	413
06500	289	13000	513
07000	369	13500	503

Fig. 21. Satisfiability benchmark.

Features	Time (ms.)	Products
50	6	48
60	25	108
70	15	104
80	8	31
90	38	0
100	55	1404
110	13	12
120	2139	1
130	511	24802
140	7	6
150	786126	1312848
160	136	5670
170	42	398
180	744	6384
190	1390	7232
200	960000	-
210	97770	800544
220	263	51
230	47	8
240	65	29
250	191	5920
260	205	7296
270	250	4301
280	960000	-
290	65	3
300	960000	-

Fig. 22. Denotational benchmark.

10. Conclusions and future work

In this paper we presented SPLA as a general framework to represent SPLs, showing how it can be used to provide FODA diagrams with a formal semantics. This semantics is fully abstract, and so any other operator that could be used to define an SPL can be represented in terms of SPLA. This suggest that the formalism is quite general and we anticipate that it can be used to express any other formalism like those mentioned in [41]. In particular, Theorem 2 showed that we work with a fully abstract model. We defined three different semantics for SPLA: an operational semantics, a denotational semantics and an axiomatic semantics. We proved that the semantics are equivalent. Besides defining the formal framework, we have developed a tool to show the applicability of our formal techniques. The tool is available at http://simba.fdi.ucm.es/at, and it has been developed under the GPL v3 license.

Since SPLA is based on process algebras, we plan to take advantage of the previous work in this field. In particular, we plan to study alternative semantics. For instance, in our current semantics the products are only sets of features. That is, the order in which the features are computed is not important. There are situations where this is no longer true. That could be the case if we consider the cost associated with the production of a product. For instance producing feature A and then B could have cost 3.00 €, but producing B and then A might cost 2.00 €. We also plan to include non-functional aspects to SPLA such as probabilities, that could be useful in building a user model [42], timed characteristics and, as we have mentioned, costs.

Another interesting feature that has been deeply studied in the field of process algebras is the probabilistic information. We think that this is an interesting feature that should studied in the world of Software Product Lines. In this way, we can deduce the probability of the products. This can be applied, for example, in software testing, so that we can add more resources to test the products with higher probabilities.

Acknowledgements

We thank the anonymous reviewers of the paper for the careful reading and many comments that have notoriously improved the final version of the paper. We also thank Professor Robert M. Hierons for his valuable comments on the previous revisions of the paper.

Appendix A. Proofs of the results

Proof Lemma 1. The proof is done by applying induction on the derivation of $P \overset{\checkmark}{\rightarrow} Q$. We have to observe that in all rules producing transitions like $P \overset{\checkmark}{\rightarrow} Q$ we see that Q is $\checkmark$. □

Proof Lemma 2. We prove this by induction on the length of the deduction of $P \overset{\checkmark}{\rightarrow} \mathtt{nil}$, let us consider that this length is n.

- $n = 0$ In this case we have two cases: $P = \checkmark$ and $P = \overline{\mathtt{A}};P$, in both cases $\emptyset \in [\![P]\!]$.
- $n > 1$ In this case we have to analyze the rules that yields $P \overset{\checkmark}{\rightarrow} \mathtt{nil}$. These rules are [**cho**1],[**cho**2], [**con**3],[**req**3], [**excl**4], and [**forb**2].

Rules [**cho1**] **and** [**cho2**]. These rules are symmetric so we can concentrate in one of them, for instance [**cho**1]. In this case $P = P_1 \vee P_2$ and $P_1 \overset{\checkmark}{\rightarrow} \mathtt{nil}$. By induction $\emptyset \in [\![P_1]\!]$, then, by Definition 7, $\emptyset \in [\![P]\!]$.

Rule [**con3**]. In this case $P = P_1 \wedge P_2$, $P_1 \overset{\checkmark}{\rightarrow} \mathtt{nil}$, and $P_2 \overset{\checkmark}{\rightarrow} \mathtt{nil}$. By induction $\emptyset \in [\![P_1]\!]$ and $\emptyset \in [\![P_2]\!]$. We have the result by Definition 7.

Rule [**req3**]. In this case $P = \mathtt{A} \Rightarrow \mathtt{Bin}P_1$ and $P_1 \overset{\checkmark}{\rightarrow} \mathtt{nil}$. By induction $\emptyset \in [\![P_1]\!]$. Since $\mathtt{A} \notin \emptyset$, by Definition 7, $\emptyset \in [\![P]\!]$.

Rule [**excl4**]. In this case $P = \mathtt{A} \not\Rightarrow \mathtt{Bin}P_1$ and $P_1 \overset{\checkmark}{\rightarrow} \mathtt{nil}$. By induction $\emptyset \in [\![P_1]\!]$. Since $\mathtt{A} \notin \emptyset$ and $\mathtt{B} \notin \emptyset$, by Definition 7, $\emptyset \in [\![P]\!]$.

Rule [**req3**]. In this case $P = P_1 \backslash \mathtt{A}$ and $P_1 \overset{\checkmark}{\rightarrow} \mathtt{nil}$. By induction $\emptyset \in [\![P_1]\!]$. Since $\mathtt{A} \notin \emptyset$, by Definition 7, $\emptyset \in [\![P]\!]$. □

Proof (*Lemma 3.1*). Since the only rule of the operational semantics that can be applied to $\mathtt{A};P$ is [**feat**], we obtain $\mathtt{tr}(\mathtt{A};P) = \{\mathtt{A} \cdot s | s \in \mathtt{tr}(P)\}$.

Thus $\mathtt{prod}(\mathtt{A};P) = \{\mathtt{A} \cup p | p \in \mathtt{prod}(P)\}$, that is the definition of $[\![\mathtt{A};]\!](\mathtt{prod}(P))$ (see Definition 7). □

Proof (*Lemma 3.2*). The rules that can be applied to $\overline{\mathtt{A}};P$ are [**ofeat**1] and [**ofeat**2]. So we obtain $\mathtt{tr}(\mathtt{A};P) = \{\swarrow\} \cup \{\mathtt{A}\cdot s | s \in \mathtt{tr}(P)\}$. Therefore $\mathtt{prod}(\overline{\mathtt{A}};P) = \{\emptyset\} \cup \{\mathtt{A} \cup p | p \in \mathtt{prod}(P)\}$, that is the definition of $[\![\overline{\mathtt{A}};]\!](\mathtt{prod}(P))$ (see Definition 7). □

Proof (*Lemma 3.3*). From rules [**cho**1] and [**cho**2] we obtain $\mathtt{tr}(P \vee P') = \mathtt{tr}(P) \cup \mathtt{tr}(P')$. Thus

$$\mathtt{prod}(P \vee P') = \mathtt{prod}(P) \cup \mathtt{prod}(P') = [\![\vee]\!](\mathtt{prod}(P), \mathtt{prod}(P')) \quad \square$$

Proof (*Lemma 3.4*). First, let us consider $[s] \in \mathtt{prod}(P \wedge P')$, we will show that $[s] \in [\![\wedge]\!](\mathtt{prod}(P),\mathtt{prod}(P'))$ by induction on the length of s.

- $s = \epsilon$. In this case $[s] = \emptyset$ and $P \wedge P' \overset{\swarrow}{\rightarrow} \mathtt{nil}$. We can only apply rule [**con**3] to obtain this transition. So $P \overset{\swarrow}{\rightarrow} \mathtt{nil}$ and $P' \overset{\swarrow}{\rightarrow} \mathtt{nil}$. Thus $\emptyset \in \mathtt{prod}(P)$ and $\emptyset \in \mathtt{prod}(P')$. Then $[s] = \emptyset \in [\![\wedge]\!](\mathtt{prod}(P),\mathtt{prod}(P'))$ by Definition 7.
- $s = \mathtt{A}\cdot s'$. In this case $P \wedge P' \overset{\mathtt{A}}{\rightarrow} P''$. We can apply rules [**con**1] or [**con**2] to obtain this transition. [**con**2] is symmetric to [**con**1], so we can concentrate on [**con**1]. P_1 exists so that $P \overset{\mathtt{A}}{\rightarrow} P_1, P'' = P_1 \wedge P'$ and $[s'] \in \mathtt{prod}(P_1 \wedge P')$. By induction $[s'] \in [\![\wedge]\!](\mathtt{prod}(P_1), \mathtt{prod}(P'))$, so there are products $p_1 \in \mathtt{prod}(P_1)$ and $p_2 \in \mathtt{prod}(P')$ such that $[s'] = p_1 \cup p_2$. Let us consider the trace $s_1 \in \mathtt{tr}(P_1)$ such that $[s_1] = p_1$. Then

$$[s] = \{\mathtt{A}\} \cup [s'] = \{\mathtt{A}\} \cup p_1 \cup p_2 = (\{\mathtt{A}\} \cup [s_1]) \cup p_2$$

Finally we obtain the result by Definition 7 since $\mathtt{A}\cdot s_1 \in \mathtt{tr}(P)$ and $p_2 \in \mathtt{prod}(P')$.

Now let us consider $p \in [\![\wedge]\!](\mathtt{prod}(P), \mathtt{prod}(P'))$. By Definition 7 there are $p_1 \in \mathtt{prod}(P)$ and $p_2 \in \mathtt{prod}(P')$ such that $p = p_1 \cup p_2$. So there are successful traces $s_1 \in \mathtt{tr}(P)$ and $s_2 \in \mathtt{tr}(P')$ such that $[s_1] = p_1$ and $[s_2] = p_2$. We make the proof by induction on the sum of the lengths of s_1 and s_2.

- $len(s_1) + len(s_2) = 0$. So $s_1 = \epsilon, s_2 = \epsilon$ and $p = \emptyset$. In this case we have the transitions $P \overset{\swarrow}{\rightarrow} \mathtt{nil}$ and $P' \overset{\swarrow}{\rightarrow} \mathtt{nil}$. By applying rule [**con**5], we have the transition $P \wedge P' \overset{\swarrow}{\rightarrow} \mathtt{nil}$. Thus $\emptyset \in \mathtt{prod}(P \wedge P')$.
- $len(s_1) + len(s_2) > 0$. Let us suppose that $s_1 = \mathtt{A}\cdot s_1'$, (the case $s_2 = \mathtt{A}\cdot s_2$ is symmetric). Then, there is a transition $P \overset{\mathtt{A}}{\rightarrow} P_1$ such that s_1' is a successful trace of P_1. By Definition 7, $[s_1'] \cup [s_2] \in [\![\wedge]\!](\mathtt{prod}(P_1), \mathtt{prod}(P'))$. By induction $[s_1'] \cup [s_2] \in \mathtt{prod}(P_1 \wedge P')$. By applying rule [**con**1], we obtain the transition $P \wedge P' \overset{\mathtt{A}}{\rightarrow} P_1 \wedge P'$. Then $\{\mathtt{A}\} \cup [s_1'] \cup [s_2] \in \mathtt{prod}(P \wedge P')$. We have the desired result since $[s_1] = \{\mathtt{A}\} \cup [s_1']$. □

Proof (*Lemma 3.5*). First we are going to prove

$$\mathtt{prod}(P \Rightarrow \mathtt{A}) \subseteq [\![\cdot \Rightarrow \mathtt{A}]\!](\mathtt{prod}(P))$$

So let us consider $p \in \mathtt{prod}(P \Rightarrow \mathtt{A})$. A successful trace $s \in \mathtt{tr}(P \Rightarrow \mathtt{A})$ exists so that $[s] = p$. We will show that $p \in [\![\cdot \Rightarrow \mathtt{A}]\!](\mathtt{prod}(P))$ by induction on the length of s. Because of Rule [**mand1**], $s \neq \epsilon$. Thus the base case is when $s = \mathtt{A}$ ($p = \{\mathtt{A}\}$). Also because of Rule [**mand1**] we have the transition $P \overset{\swarrow}{\rightarrow} \mathtt{nil}$, so $\emptyset \in \mathtt{prod}(P)$. Therefore, by Definition 7 $p = \{\mathtt{A}\} \cup \emptyset \in [\![\cdot \Rightarrow \mathtt{A}]\!](\mathtt{prod}(P))$.

Now let us consider $s = \mathtt{B}\cdot s'$. If $\mathtt{A} = \mathtt{B}$, we have the transition $P \Rightarrow \mathtt{A} \overset{A}{\rightarrow} P_1$ and $s' \in \mathtt{tr}(P_1)$. This transition can only be deduced by Rule [**mand2**]. Therefore $P \overset{A}{\rightarrow} P_1$ and $s \in \mathtt{tr}(P)$. To obtain the result it is only necessary to take into account that $\mathtt{A} \in [s]$, so $[s] = \{\mathtt{A}\} \cup [s]$. If $\mathtt{A} \neq \mathtt{B}$, we have the transition $P \Rightarrow \mathtt{A} \overset{B}{\rightarrow} P_1 \Rightarrow \mathtt{A}$ and $s' \in \mathtt{tr}(P_1 \Rightarrow \mathtt{A})$. This transition can only be deduced by Rule [**mand3**]. Therefore $P \overset{B}{\rightarrow} P_1$. By induction we have $[s'] \in [\![\cdot \Rightarrow \mathtt{A}]\!](\mathtt{prod}(P_1))$. Thus, by Definition 7, there is $p \in \mathtt{prod}(P_1)$ such that $p_1 \cup \{\mathtt{A}\} = [s']$. There must exist $s_1 \in \mathtt{tr}(P_1)$ such that $[s_1] = p_1$. Now we can observe that $\mathtt{B}\cdot s_1 \in \mathtt{tr}(P)$, thus $\{\mathtt{B}\} \cup p_1 \in \mathtt{prod}(P)$. Thus

$$p = [s] = \{\mathtt{B}\} \cup [s'] = \{\mathtt{B}\} \cup p_1 \cup \{\mathtt{A}\} \in [\![\cdot \Rightarrow \mathtt{A}]\!](\mathtt{prod}(P))$$

Now we are going to prove

$$[\![\cdot \Rightarrow \mathtt{A}]\!](\mathtt{prod}(P)) \subseteq \mathtt{prod}(P \Rightarrow \mathtt{A})$$

So let us consider $p \in [\![\cdot \Rightarrow \mathtt{A}]\!](\mathtt{prod}(P))$. Then, by Definition 7, there is $p' \in \mathtt{prod}(P)$ such that $p = p' \cup \{\mathtt{A}\}$. There exists $s \in \mathtt{tr}(P)$ such that $[s] = p'$. We will show that $p \in \mathtt{prod}(P \Rightarrow \mathtt{A})$ by induction on the length of s.

- $s = \epsilon$. In this case we have $P \overset{\swarrow}{\rightarrow} \mathtt{nil}$ and $p' = \emptyset$. Then by applying Rules [**mand1**] and [**tick**], we obtain

$$P \Rightarrow \mathtt{A} \overset{A}{\rightarrow} \swarrow \overset{\swarrow}{\rightarrow} \mathtt{nil}$$

Then $p = p' \cup \{\mathtt{A}\} = \{\mathtt{A}\} \in \mathtt{prod}(P \Rightarrow \mathtt{A})$.
- $s = \mathtt{A}\cdot s'$ In this case we have the transition $P \overset{\mathtt{A}}{\rightarrow} P'$ and $s' \in \mathtt{tr}(P')$. Then, by Rule [**mand2**], we obtain $P \Rightarrow \mathtt{A} \overset{\mathtt{A}}{\rightarrow} P'$. Therefore $s \in \mathtt{tr}(P \Rightarrow \mathtt{A})$. To get the result it is enough to take into account that $\mathtt{A} \in [s]$, so $p = p' \cup \{\mathtt{A}\} = [s] \cup \{\mathtt{A}\} = [s]$.
- $s = \mathtt{B}\cdot s'$ with $\mathtt{A} \neq \mathtt{B}$. In this case we have $P \overset{\mathtt{B}}{\rightarrow} P'$ and $s' \in \mathtt{tr}(P')$. By Definition 7,

$$\{\mathtt{A}\} \cup [s'] \in [\![\cdot \Rightarrow \mathtt{A}]\!](\mathtt{prod}(P'))$$

By induction we get

$$\{\mathtt{A}\} \cup [s'] \in \mathtt{prod}(P' \Rightarrow \mathtt{A})$$

There is a trace $s'' \in \mathtt{tr}(P' \Rightarrow \mathtt{A})$ such that $[s''] = \{\mathtt{A}\} \cup [s']$. By applying Rule [**mand3**], $P \Rightarrow \mathtt{A} \overset{\mathtt{B}}{\rightarrow} P' \Rightarrow \mathtt{A}$. Finally, since $\mathtt{B}\cdot s'' \in \mathtt{tr}(P \Rightarrow \mathtt{A})$, we obtain

$$p = p' \cup \{\mathtt{A}\} = [s] \cup \{\mathtt{A}\} = \{\mathtt{B}\} \cup [s'] \cup \{\mathtt{A}\} = \{\mathtt{B}\} \cup [s''] \in \mathtt{prod}(P \Rightarrow \mathtt{A}) \quad \square$$

Proof (*Lemma 3.6*). First we are going to prove

$$\mathtt{prod}(P \setminus \mathtt{A}) \subseteq [\![\cdot \setminus \mathtt{A}]\!](\mathtt{prod}(P))$$

So let us consider $p \in \mathtt{prod}(P\setminus\mathtt{A})$. A successful trace $s \in \mathtt{tr}(P\setminus\mathtt{A})$ exists such that $[s] = p$. We will show that $p \in [\![\cdot \setminus \mathtt{A}]\!](\mathtt{prod}(P))$ by induction on the length of s.

- $s = \epsilon$ In this case we have $p = \emptyset$ and $P \setminus \mathtt{A} \overset{\swarrow}{\rightarrow} \mathtt{nil}$. The only applicable rule is Rule [**forb2**], thus $P \overset{\swarrow}{\rightarrow} \mathtt{nil}$. By Lemma 2, $\emptyset \in \mathtt{prod}(P)$. By Definition 7 $p = \emptyset \in \mathtt{prod}(P\setminus\mathtt{A})$.
- $s = \mathtt{B}\cdot s'$ In this case $P \setminus \mathtt{A} \overset{B}{\rightarrow} P' \setminus \mathtt{A}$ and $s' \in \mathtt{tr}(P'\setminus\mathtt{A})$. Since s is successful, s' is also successful. Thus $[s'] \in \mathtt{prod}(P'\setminus\mathtt{A})$. By induction $[s'] \in [\![\cdot \setminus \mathtt{A}]\!](P')$, therefore $[s'] \in \mathtt{prod}(P')$ and $\mathtt{A} \notin [s']$. Since the only applicable rule to deduce $P \setminus \mathtt{A} \overset{B}{\rightarrow} P' \setminus \mathtt{A}$ is Rule [**forb1**], we obtain $\mathtt{B}\cdot s' \in \mathtt{tr}(P)$ and $\mathtt{B} \neq \mathtt{A}$. Therefore $[s] \in \mathtt{tr}(P)$ and $\mathtt{A} \notin [s]$, so, by Definition 7, $p = [s] \in [\![\cdot \setminus \mathtt{A}]\!](\mathtt{prod}(P))$.

Now we are going to prove

$[\![\cdot \backslash \mathtt{A}]\!](\mathtt{prod}(P)) \subseteq \mathtt{prod}(P \setminus \mathtt{A})$

So let us consider $p \in [\![\cdot \backslash \mathtt{A}]\!](\mathtt{prod}(P))$. By Definition 7 $p \in \mathtt{prod}(P)$ and $\mathtt{A} \notin p$. Therefore there exists a successful trace $s \in \mathtt{tr}(P)$ such that $[s] = p$. Since $\mathtt{A} \notin [s]$, we obtain that s is a successful trace of $P \backslash \mathtt{A}$. Therefore $p = [s] \in \mathtt{prod}(P \backslash \mathtt{A})$. □

Proof (*Lemma 3.7*). First we are going to prove

$\mathtt{prod}(\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P) \subseteq [\![\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \cdot]\!](\mathtt{prod}(P))$

So let us consider $p \in \mathtt{prod}(\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}\,P)$. There exists a successful trace $s \in \mathtt{tr}(\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}P)$ such that $[s] = p$. We will show that $p \in [\![\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in} \cdot]\!](\mathtt{prod}(P))$ by induction on the length of s.

$s = \epsilon$. In this case we have the transition

$\mathtt{prod}(\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P) \xrightarrow{\swarrow} \mathtt{nil}$

The only applicable rule is Rule [**req3**]. So $P \xrightarrow{\swarrow} \mathtt{nil}$ and therefore $\emptyset \in \mathtt{prod}(P)$. So by Definition 7, $\emptyset \in [\![\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in} \cdot]\!](\mathtt{prod}(P))$.

$s = \mathtt{C} \cdot s'$ with $\mathtt{C} \neq \mathtt{A}$. In this case we have the transition $\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P \xrightarrow{\mathtt{C}} P'$. If $\mathtt{A} \neq \mathtt{C}$, by Rule [**req1**], we obtain $P' = \mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in} P_1$ with $P \xrightarrow{\mathtt{C}} P_1$, and $s' \in \mathtt{tr}(\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in} P_1)$ By induction, we obtain $[s'] \in [\![\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}\ \cdot]\!](\mathtt{prod}(P_1))$. By Definition 7, we have two cases

$\mathtt{A} \notin [s']$. In this case $[s'] \in \mathtt{prod}(P_1)$, therefore $\{\mathtt{C}\} \cup [s'] \in \mathtt{prod}(P)$ and, since $\mathtt{C} \neq \mathtt{A}$ by Definition 7, $\{\mathtt{C}\} \cup [s'] \in [\![\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in} \cdot]\!](\mathtt{prod}(P))$

$\mathtt{A} \in [s']$. In this case, by Definition 7, there is a trace $s'' \in \mathtt{tr}(P_1)$ such that $[s'] = [s''] \cup \{\mathtt{B}\}$. By Definition 7, $\{\mathtt{C}\} \cup [s'] \cup \{\mathtt{B}\} \in [\![\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in} \cdot]\!](\mathtt{prod}(P))$.

$s = \mathtt{A} \cdot s'$. In this case we have the transition

$\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P \xrightarrow{\mathtt{A}} P'$

Since the only applicable rule is Rule [**req2**], we obtain that there is $P_1 \in \mathtt{SPLA}$ such that $P \xrightarrow{\mathtt{A}} P_1$ $P' = P_1 \Rightarrow \mathtt{B}$ and $s' \in \mathtt{tr}(P_1 \Rightarrow \mathtt{B})$. By Lemma 5, $[s'] \in [\![\ \cdot \Rightarrow \mathtt{B}]\!]\mathtt{prod}(P_1)$. By Definition 7, there is $s'' \in \mathtt{tr}(P_1)$ such that $[s'] = [s''] \cup \{\mathtt{B}\}$. Since $P \xrightarrow{\mathtt{A}} P_1, \mathtt{A} \cdot s'' \in \mathtt{tr}(P)$. Therefore, by Definition 7,

$p = [s] = \{\mathtt{A}\} \cup [s'] = \{\mathtt{A}\} \cup [s''] \cup \{\mathtt{B}\} \in [\![\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \cdot]\!](\mathtt{prod}(P))$

Now we are going to prove

$[\![\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \cdot]\!](\mathtt{prod}(P)) \subseteq \mathtt{prod}(\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P)$

So let us consider $p \in [\![\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}\ \cdot]\!](\mathtt{prod}(P))$. By Definition 7 there are two cases:

- $\mathtt{A} \notin p$. In this case $p \in \mathtt{prod}(P)$, so there is $s \in \mathtt{tr}(P)$ such that $[s] = p$. Since $\mathtt{A} \notin P$, by applying rule [**req1**], $s \in \mathtt{tr}(\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}P)$.
- There is $p' \in \mathtt{prod}(P)$ such that $\mathtt{A} \in p'$ and $p = p' \cup \{\mathtt{B}\}$. Let us consider $s \in \mathtt{tr}(P)$ such that $p' = [s]$. So, there are traces s_1 and s_2 such that $s = s_1 \cdot \mathtt{A} \cdot s_2$ and $\mathtt{A} \notin s_1$. So, there exists P_1 and P_2 such that $P \xrightarrow{s_1} P_1 \xrightarrow{A} P_2$ and $s_2 \in \mathtt{tr}(P_2)$. Then, by applying rule [**req1**],

$\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P \xrightarrow{s_1} \mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P_1 \xrightarrow{\mathtt{A}} P_2 \Rightarrow \mathtt{B}$

Since $[s_2] \in \mathtt{prod}(P_2)$, by Lemma 5, $[s_2] \cup \{\mathtt{B}\} \in \mathtt{prod}(P_2 \Rightarrow \mathtt{B})$. Let us consider $s_2' \in \mathtt{tr}(P_2 \Rightarrow \mathtt{B})$ such that $[s_2'] = [s_2] \cup \{\mathtt{B}\}$. Therefore $s_1 \cdot \mathtt{A} \cdot s_2' \in \mathtt{tr}(\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P)$, so

$p = p' \cup \{\mathtt{B}\} = [s_1] \cup \{\mathtt{A}\} \cup [s_2] \cup \{\mathtt{B}\} = [s_1] \cup \{\mathtt{A}\} \cup [s_2'] \in \mathtt{prod}(\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P)$ □

Proof (*Lemma 3.8*). First we are going to prove

$\mathtt{prod}(\mathtt{A} \Rrightarrow \mathtt{B}\ \mathtt{in}\ P) \subseteq [\![\mathtt{A} \Rrightarrow \mathtt{B}\ \mathtt{in}\ \cdot]\!](\mathtt{prod}(P))$

So let us consider $p \in \mathtt{prod}(\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}P)$. There exists a successful trace $s \in \mathtt{tr}(\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}P)$ such that $[s] = p$. We will show that $p \in [\![\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in} \cdot]\!](\mathtt{prod}(P))$ by induction on the length of s.

$s = \epsilon$. In this case we have the transition

$\mathtt{prod}(\mathtt{A} \Rrightarrow \mathtt{B}\ \mathtt{in}\ P) \xrightarrow{\swarrow} \mathtt{nil}$

The only applicable rule is Rule [**excl4**]. So $P \xrightarrow{\swarrow} \mathtt{nil}$ and therefore $\emptyset \in \mathtt{prod}(P)$. So by Definition 7, we have $\emptyset \in [\![\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in} \cdot]\!](\mathtt{prod}(P))$.

$s = \mathtt{C} \cdot s'$ with $\mathtt{C} \neq \mathtt{A}$ and $\mathtt{C} \neq \mathtt{B}$. In this case we have the transition $\mathtt{A} \Rrightarrow \mathtt{B}\ \mathtt{in}\ P \xrightarrow{\mathtt{C}} P'$. If $\mathtt{A} \neq \mathtt{C}$, because of Rule [**excl1**], we obtain $P' = \mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in} P_1$ with $P \xrightarrow{\mathtt{C}} P_1$, and $s' \in \mathtt{tr}(\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in} P_1)$. By induction, we obtain $[s'] \in [\![\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}\ \cdot]\!](\mathtt{prod}(P_1))$. By Definition 7 there are two cases:

- $\mathtt{A} \notin [s']$. In this case $[s'] \in \mathtt{prod}(P_1)$. Therefore $\{\mathtt{C}\} \cup [s'] \in \mathtt{prod}(P)$. Since $\mathtt{A} \notin \{\mathtt{C}\} \cup [s']$,

$p = \{\mathtt{C}\} \cup [s'] \in [\![\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}\ \cdot]\!](\mathtt{prod}(P))$

- $\mathtt{B} \notin [s']$. This case is identical to the previous one.

Now let us prove

$[\![\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}\ \cdot]\!](\mathtt{prod}(P)) \subseteq \mathtt{prod}(\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}\,P)$

Let us consider $p \in [\![\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in} \cdot]\!](\mathtt{prod}(P))$. There are two cases

$\mathtt{A} \notin p$. In this case $p \in \mathtt{prod}(P)$. So there exists a trace $s \in \mathtt{tr}(P)$ such that $p = [s]$. We prove this by induction on the length of s.

$s = \epsilon$. In this case we have the transition $P \xrightarrow{\swarrow} \mathtt{nil}$. Then by Rule [**excl4**], $\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}\,P \xrightarrow{\swarrow} \mathtt{nil}$. So $p = \emptyset \in \mathtt{prod}(\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}P)$

$s = \mathtt{C} \cdot s'$. Now we have two possibilities: $\mathtt{C} = \mathtt{B}$ or $\mathtt{C} \neq \mathtt{B}$. In the first case we have the transition $P \xrightarrow{\mathtt{B}} P_1$ with $s' \in \mathtt{tr}(P_1)$. By Rule [**excl3**], we obtain the transition $\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}\,P \xrightarrow{\mathtt{B}} P_1 \setminus \mathtt{A}$. On the one hand $[s'] \in \mathtt{prod}(P_1)$ and on the other hand $\mathtt{A} \notin [s'] \subseteq p$, so $[s'] \in [\![\cdot \backslash \mathtt{A}]\!](P_1)$. Because of Lemma 6 $[s'] \in \mathtt{prod}(P_1 \backslash \mathtt{A})$, and then

$p = \{\mathtt{C}\} \cup [s'] = \{\mathtt{B}\} \cup [s'] \in \mathtt{prod}(\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}\,P)$

If $\mathtt{C} \neq \mathtt{B}$ we have the transition $P \xrightarrow{\mathtt{C}} P_1$ with $s' \in \mathtt{tr}(P_1)$. By Rule [**excl1**], we obtain the transition $\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}\,P \xrightarrow{\mathtt{C}} \mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}\,P_1$. Since $\mathtt{A} \notin [s']$, we obtain $[s'] \in [\![\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in} \cdot]\!](\mathtt{prod}(P_1))$ By induction we obtain $[s'] \in \mathtt{prod}(\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}P_1)$, so

$p = \{\mathtt{C}\} \cup [s'] \in \mathtt{prod}(\mathtt{A} \Rrightarrow \mathtt{B}\,\mathtt{in}\,P)$

$\mathtt{B} \notin p$. This case is similar to the previous one. The only difference appears in the inductive case when making the possibilities of $\mathtt{C}$. In this case the possibilities are: $\mathtt{C} = \mathtt{A}$ and $\mathtt{C} \neq \mathtt{A}$. The second possibility is like the second possibility in the previous case. The difference with respect to the first possibility is that we need to apply Rule [**excl2**] instead of Rule [**excl3**]. □

Proof (Theorem 1). We prove this by using structural induction on P taking into account the previous lemmas. The base cases are $P = \mathtt{nil}$ and $P = \blacktriangleright$. In this case it is easy to check that $\mathtt{prod}(\mathtt{nil}) = [\![\mathtt{nil}]\!] = \emptyset$ and $\mathtt{prod}(\blacktriangleright) = [\![\blacktriangleright]\!] = \{\emptyset\}$. The induction corresponds to

$P = \mathtt{A};P'$ This case corresponds to Lemma 3.1.
$P = \overline{\mathtt{A}};P'$ This case corresponds to Lemma 3.2.
$P = P_1 \vee P_2$ This case corresponds to Lemma 3.3.
$P = P_1 \wedge P_2$ This case corresponds to Lemma 3.4.
$P = \mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}\,P$ This case corresponds to Lemma 3.7.
$P = \mathtt{A} \not\Rightarrow \mathtt{B}\,\mathtt{in}\,P$ This case corresponds to Lemma 3.8.
$P = P\backslash\mathtt{A}$ This case corresponds to Lemma 3.6.
$P = P \Rightarrow \mathtt{A}$ This case corresponds to Lemma 3.5. □

Proof (Corollary 1). We just need to consider the following:

$$\mathtt{prod}(\mathtt{op}(P_1,\ldots,P_n)) = [\![\mathtt{op}(P_1,\ldots,P_n)]\!] = [\![\mathtt{op}]\!]([\![P_1]\!],\ldots,[\![P_n]\!])$$

Since $P_i \equiv Q_i$, $[\![P_i]\!] = \mathtt{prod}(P_i) = \mathtt{prod}(Q_i) = [\![Q_i]\!]$. Thus

$$[\![\mathtt{op}]\!]([\![P_1]\!],\ldots,[\![P_n]\!]) = [\![\mathtt{op}]\!]([\![Q_1]\!],\ldots,[\![Q_n]\!]) = [\![\mathtt{op}(Q_1,\ldots,Q_n)]\!] = \mathtt{prod}(\mathtt{op}(Q_1,\ldots,Q_n)) \quad \square$$

Proof (Proposition 9). Let us consider $\mathtt{A},\mathtt{B},\mathtt{C},\mathtt{D} \in \mathcal{F}$. If $\mathtt{A} \neq \mathtt{D}$ it is easy to check, by using the definition of the *require* operator in Definition 7, the following

$$[\![\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{C} \Rightarrow \mathtt{D}\ \mathtt{in}\ P]\!] = [\![\mathtt{C} \Rightarrow \mathtt{D}\ \mathtt{in}\ \mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P]\!]$$

Instead if $\mathtt{A} = \mathtt{D}$, because the term is closed under the *require* constraint, again by using the definition of the *require* operator:

$$\begin{aligned}
&[\![\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{C} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{C} \Rightarrow \mathtt{A}\ \mathtt{in}\ P]\!] =\\
&[\![\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{C} \Rightarrow \mathtt{A}\ \mathtt{in}\ \mathtt{C} \Rightarrow \mathtt{B}\ \mathtt{in}\ P]\!] =\\
&[\![\mathtt{C} \Rightarrow \mathtt{A}\ \mathtt{in}\ \mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{C} \Rightarrow \mathtt{B}\ \mathtt{in}\ P]\!] =\\
&[\![\mathtt{C} \Rightarrow \mathtt{A}\ \mathtt{in}\ \mathtt{C} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P]\!] =\\
&[\![\mathtt{C} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{C} \Rightarrow \mathtt{A}\ \mathtt{in}\ P]\!] =\\
&\quad [\![\mathtt{C} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{C} \Rightarrow \mathtt{A}\ \mathtt{in}\ \mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P]\!]
\end{aligned}$$

Let us prove in detail

$$[\![\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{C} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{C} \Rightarrow \mathtt{A}\ \mathtt{in}\ P]\!] \subseteq [\![\mathtt{C} \Rightarrow \mathtt{B}\ \mathtt{in}\ \mathtt{C} \Rightarrow \mathtt{A}\ \mathtt{in}\ \mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ P]\!]$$

the other cases are similar. Any product $p \in [\![\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}\,\mathtt{C} \Rightarrow \mathtt{B}\,\mathtt{in}\,\mathtt{C} \Rightarrow \mathtt{A}\,\mathtt{in}\,P]\!]$ is built from a product $p' \in [\![P]\!]$. We can distinguish the following cases:

$\mathtt{C} \in p'$. Then $p = p' \cup \{\mathtt{A},\mathtt{B}\}$ and $p \in [\![\mathtt{C} \Rightarrow \mathtt{B}\,\mathtt{in}\,\mathtt{C} \Rightarrow \mathtt{A}\,\mathtt{in}\,\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}\,P]\!]$.
$\mathtt{C} \notin p', \mathtt{A} \in p'$. Then $p = p' \cup \{\mathtt{B}\}$ and $p \in [\![\mathtt{C} \Rightarrow \mathtt{B}\,\mathtt{in}\,\mathtt{C} \Rightarrow \mathtt{A}\,\mathtt{in}\,\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}\,P]\!]$.
$\mathtt{C},\mathtt{A} \notin p'$. Then $p = p'$ and $p \in [\![\mathtt{C} \Rightarrow \mathtt{B}\,\mathtt{in}\,\mathtt{C} \Rightarrow \mathtt{A}\,\mathtt{in}\,\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}\,P]\!]$. □

Proof (Proposition 4). This proposition is immediate by the definition of the *exclude* operator in Definition 7. □

Proof (Theorem 2). Since $\mathcal{F}$ is finite A is finite, so we can prove the result by induction on $|A|$.

$|A| = 0$ In this case the $P = \mathtt{nil}$.

$|A| > 0$ Let us consider a product $p \in A$ and the set $A' = A\backslash\{p\}$. By induction there is $P' \in \mathtt{SPLA}_b$ such that $\mathtt{prod}(P') = A'$. Since $\mathcal{F}$ is finite, p is a finite set of features $p = \{\mathtt{A}_1,\ldots\mathtt{A}_n\}$. Then the term

$$P = (\mathtt{A}_1;\ldots\mathtt{A}_n;\blacktriangleright) \vee P'$$

satisfies the thesis of the result. □

Proof (Proposition 5). In all axioms but [**PRE1**], [**PRE4**], [**PRE5**], [**REQ3**], [**CON1**], [**CON4**] the transitions of the terms on both sides of the equation are the same. So in these cases the traces are the same, so they have the same products.

In the other cases the proof is done by using the denotational semantics.

[**PRE1**] $\mathtt{A};\mathtt{B};P =_E \mathtt{B};\mathtt{A};P$

$$[\![\mathtt{A};\mathtt{B};P]\!] = \{\mathtt{A}\} \cup [\![\mathtt{B};P]\!] = \{\mathtt{A}\} \cup \{\mathtt{B}\} \cup [\![P]\!] = \{\mathtt{B}\} \cup \{\mathtt{A}\} \cup [\![P]\!] = \{\mathtt{B}\} \cup [\![\mathtt{A};P]\!] = [\![\mathtt{B};\mathtt{A};P]\!]$$

[**PRE4**] $\mathtt{A};\mathtt{nil} =_E \mathtt{nil}$

$$p \in [\![\mathtt{A};\mathtt{nil}]\!] \Longleftrightarrow \exists p' \in [\![\mathtt{nil}]\!] \wedge p = p' \cup \{\mathtt{A}\} \Longleftrightarrow \text{False} \Longleftrightarrow p \in [\![\mathtt{nil}]\!]$$

[**PRE5**] $\mathtt{A};\mathtt{A};P =_E \mathtt{A};P$

$$[\![\mathtt{A};\mathtt{A};P]\!] = \{\mathtt{A}\} \cup [\![\mathtt{A};P]\!] = \{\mathtt{A}\} \cup \{\mathtt{A}\} \cup [\![P]\!] = \{\mathtt{A}\} \cup [\![P]\!] = [\![\mathtt{A};P]\!]$$

[**REQ3**] $\mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}\,(\mathtt{B};P) =_E \mathtt{B};P$. In this case it is enough to consider that $\mathtt{B} \in [\![\mathtt{B};P]\!]$. Therefore by the Definition 7,

$$p \in [\![\mathtt{A} \Rightarrow \mathtt{B}\ \mathtt{in}\ \cdot\,]\!]([\![\mathtt{B};P]\!]) \Longleftrightarrow p \in [\![\mathtt{B};P]\!]$$

[**CON 1**] $(\mathtt{A};P) \wedge Q =_E \mathtt{A};(P \wedge Q)$

$$\begin{aligned}
&p \in [\![(\mathtt{A};P) \wedge Q]\!] \Longleftrightarrow\\
&\exists p_1 \in [\![\mathtt{A};P]\!],\ p_2 \in [\![Q]\!] : p = p_1 \cup p_2 \Longleftrightarrow\\
&\exists p_1' \in [\![P]\!],\ p_2 \in [\![Q]\!] : p = p_1' \cup \{\mathtt{A}\} \cup p_2 \Longleftrightarrow\\
&\exists p' \in [\![P \wedge Q]\!] : p = p' \cup \{\mathtt{A}\} \Longleftrightarrow\\
&p \in [\![\mathtt{A};(P \wedge Q)]\!]
\end{aligned}$$

[**CON 4**] $P \wedge \mathtt{nil} =_E \mathtt{nil}$

$$\begin{aligned}
&p \in [\![P \wedge \mathtt{nil}]\!] \Longleftrightarrow\\
&\exists p_1 \in [\![P]\!], p_2 \in [\![\mathtt{nil}]\!] : p = p_1 \cup p_2 \Longleftrightarrow\\
&\exists p_1 \in [\![P]\!], p_2 \in \emptyset : p = p_1 \cup p_2 \Longleftrightarrow\\
&\text{False} \Longleftrightarrow p \in [\![\mathtt{nil}]\!] \quad \square
\end{aligned}$$

Proof (Theorem 3). The proof is made trivially by structural induction on P. □

Proof Lemma 4. The proof is made by induction on the depth of P. Let us note that each P_i term appearing in part 2 of Definition 13 cannot be $\mathtt{nil}$. □

Proof (Lemma 5). The proof is made by structural induction of P by applying Axioms [**PRE 1**] and [**PRE 3**]. □

Proof Proposition 6. The proof is made by structural induction of P and applying Lemma 5. □

Proof (Proposition 7). The proof is made by induction on the depth of the pre-normal form P. The base case is trivial because $P = \lightning$ or $P = \mathtt{nil}$ and in these cases P are already in normal form.

The inductive case needs further explanations. The difference between a normal form and a pre-normal form is the ordering imposed to the features. Basically there are two cases. The first case is when P has the following form:

$$P = \ldots \vee \mathtt{B}; P \vee \ldots \vee \mathtt{A}; Q \vee \ldots \qquad \text{with } \mathtt{A} < \mathtt{B}$$

This term can be transformed into normal form by applying the commutativity of the *choose-one* operator (Eq. [**CHO 1**]).

The second case is when the features that are not properly ordered appear in the same subterm:

$$P = \ldots \vee \mathtt{B}; (\ldots \mathtt{A}; P \ldots) \vee \ldots \qquad \text{with } \mathtt{A} < \mathtt{B}$$

This case can be transformed by applying the Eqs. [**PRE 3**] and [**PRE 1**] as indicated below:

$$\begin{aligned} &\mathtt{B}; ((\mathtt{A}; P) \vee Q) =_E && \text{[PRE3]}\\ &(\mathtt{B}; \mathtt{A}; P) \vee (\mathtt{B}; Q) =_E && \text{[PRE1]}\\ &(\mathtt{A}; \mathtt{B}; P) \vee (\mathtt{B}; Q) && \square \end{aligned}$$

Proof (Proposition 8). Let us make an argument by contradiction. Let us suppose we have P, $Q \in \mathtt{SPLA}_{nf}$ and let us suppose that they are syntactically different and let us prove that they are not equivalent, $P \not\equiv Q$. The proof is made by induction on the depth of P and Q. The base case is when both are either $\lightning$ or $\mathtt{nil}$ and the result is trivial. In the inductive case we have the following possibilities:

(a) $P = (\mathtt{A}_1; P_1) \vee (\mathtt{A}_m; P_n)$,
(b) $P = (\mathtt{A}_1; P_1) \vee (\mathtt{A}_m; P_n) \vee \lightning$,
(c) $Q = (\mathtt{B}_1; Q_1) \vee (\mathtt{B}_m; Q_m)$, or
(d) $Q = (\mathtt{B}_1; Q_1) \vee (\mathtt{B}_m; Q_m) \vee \lightning$

If we are in case (a) + (d) we $\emptyset \in \mathtt{prod}(P)$ but $\notin \mathtt{prod}(Q)$, so $P \not\equiv Q$. The case (b) + (c) is symmetric. Let us consider case (a) + (c), the case (b) + (d) is solved in the same way. If P and Q are syntactically different then there are two possibilities:

- $\{\mathtt{A}_1, \ldots, \mathtt{A}_n\} \neq \{\mathtt{B}_1, \ldots, \mathtt{B}_m\}$. Let us consider the first difference among both sets. Let us assume that the first one consist in an element from the first set that is not in the second set, that is, there is $k \in \{1, \ldots, n\}$ such that $\mathtt{A}_k \notin \{\mathtt{B}_1 \ldots, \mathtt{B}_m\}$ and $\mathtt{A}_i = \mathtt{B}_i$ for $i < k$. There are the following possibilities:
  - $k = m + 1$ Any occurrence of $\mathtt{A}_k$ in Q should be in any of the sub-trees Q_j, $j < k$. By Lemma 4, that would imply that $\mathtt{B}_j \in p$ for any product $p \in \mathtt{prod}(Q)$ such that $\mathtt{A}_k \in P$. But $\mathtt{B}_j < \mathtt{A}_k$, then $\mathtt{B}_j \notin \mathtt{voc}(P_k)$. Therefore there are products $p' \in \mathtt{prod}(P)$ such that $\mathtt{B}_j \notin p'$. Therefore $P \not\equiv Q$.
  - $\mathtt{A}_k < \mathtt{B}_k$ This case is similar to the previous one because $\mathtt{A}_k \neq \mathtt{B}_l$ for $l \geqslant k$ and $\mathtt{A}_k \notin \mathtt{voc}(Q_l)$ for $l \geqslant k$.
  - $\mathtt{A}_k > \mathtt{B}_k$ This case is symmetric to the previous one.
- $n = m$, $\mathtt{A}_i = \mathtt{B}_i$ for $1 \leqslant i \leqslant n$, and $\{P_1, \ldots, P_n\} \neq \{Q_1, \ldots, Q_n\}$. Let us consider the first k such that $P_k \neq Q_k$. By structural induction we have $P_k \not\equiv Q_k$, let us assume that $p \in \mathtt{prod}(P_k)$ but $p \notin \mathtt{prod}(Q_k)$. It is clear that $p \cup \{\mathtt{A}_k\} \in \mathtt{prod}(P)$, we are going to prove that $p \cup \{\mathtt{A}_k\} \notin \mathtt{prod}(Q)$. On the one hand, since $\mathtt{A}_k \notin \mathtt{voc}(Q_i)$ for $i \geqslant k$, by Lemma 4, $p \cup \{\mathtt{A}_k\} \notin \mathtt{prod}(\mathtt{A}_i; Q_i)$ for $i \geqslant k$. On the other hand, since $\mathtt{A}_i \notin \mathtt{voc}(P_k)$ for $i < k$, again by Lemma 4, $\mathtt{A}_i \notin p$ and thus $\mathtt{A}_i \notin p \cup \{\mathtt{A}_k\}$ for $i < k$. Since $\mathtt{A}_i \in q$ for any $q \in \mathtt{prod}(\mathtt{A}_i; Q_i)$, $p \cup \{\mathtt{A}_k\} \notin \mathtt{prod}(\mathtt{A}_i; Q_i)$ for $i < k$. Thus, for any $1 \leqslant i \leqslant n, p \cup \{\mathtt{A}_k\} \notin \mathtt{prod}(\mathtt{A}_i; Q_i)$ and then $p \cup \{\mathtt{A}_k\} \notin \mathtt{prod}(Q)$. Therefore $P \not\equiv Q$. $\square$

Proof Theorem 4. We have to prove two implications

- If $P =_{E\ Q}$ then $P \equiv Q$. This is a consequence of the soundness of each rule (Proposition 5).
- If then $P \equiv Q$ then $P =_{E\ Q}$. By Proposition 6, there are P_{pre}, $Q_{pre} \in \mathtt{SPLA}_{pre}$ such that $P_{pre} =_{E\ P}$ and $Q =_E Q_{pre}$. Now by Proposition 7, there are $P_{nf}, Q_{nf} \in \mathtt{SPLA}_{nf}$ such that $P_{nf} =_E P_{pre} =_{E\ P}$ and $Q_{nf} =_{E\ Q} pre =_{E\ Q}$. By Proposition 5, we have

$$P_{nf} \equiv P_{pre} \equiv P \equiv Q \equiv Q_{pre} \equiv Q_{nf}$$

Finally, by Proposition 8, we have that P_{nf} and Q_{nf} are identical so $P =_{E\ Q}$. $\square$

Proof (Lemma 6).

1. By construction of $\varphi_1^{\Rightarrow \varphi_2}$.
2. It is enough to consider the valuation v' defined as:

$$v'(\mathtt{A}_l) = \begin{cases} v(\mathtt{A}_k) & \text{if } 0 \leqslant \mathsf{maxin}(\mathtt{A}_l, \varphi) < k \\ v(\mathtt{A}_l) & \text{otherwise} \end{cases}$$

3. By construction of $\varphi_1^{\Rightarrow \varphi_2}$. $\square$

Proof (Lemma 7). The proof will be done immediate by structural induction on P.

$P = \mathtt{nil}$ or $P = \lightning$. Trivial since $\mathsf{vars}(\phi(P)) = \emptyset$.

$P = \mathtt{B}; P'$. In this case $\phi(P) = \mathtt{B}_{m+1} \wedge \phi(P')$ where $m = \mathsf{maxin}(\mathtt{B}, \phi(P))$. In this case $\mathtt{A}_l \neq \mathtt{B}_{m+1}$. So either $\mathtt{A} \neq \mathtt{B}$ or $\mathtt{A} = \mathtt{B}$ and $l \neq m + 1$. In the second case we can deduce $l \leqslant m$, because of the definition of maxin. So in both cases $\mathtt{A}_l \in \mathsf{vars}(\phi(P'))$ so we obtain the result by structural induction.

$P = \overline{\mathtt{B}}; P'$. Trivial since $\mathsf{vars}(\phi(P)) = \emptyset$.

$P = P_1 \vee P_2$ In this case $\phi(P) = \phi(P_1)^{\Rightarrow \phi(P_2)} \vee \phi(P_1)^{\Rightarrow \phi(P_2)}$. Since $v \models \phi(P)$, then $v \models \phi(P_1)^{\Rightarrow \phi(P_2)}$ or $v \models \phi(P_2)^{\Rightarrow \phi(P_1)}$. Let assume the first case. Since $v \models \phi(P_1)^{\Rightarrow \phi(P_2)}$, $v \models \phi(P_1)$. Then we have the following cases

$\mathtt{A}_l \in \mathsf{vars}(\phi(P_1))$ Then we obtain the result by induction on P_1.

$\mathtt{A}_l \notin \mathsf{vars}(\phi(P_1))$. Let consider $m = \mathsf{maxin}(\mathtt{A}, \phi(P_1))$. Then, by Lemma 6.3, $m < l \leqslant \mathsf{maxin}(\mathtt{A}, \phi(P_2))$. Since $v(\mathtt{A}_l) = 0$ and $v \models \phi(P_2)^{\Rightarrow \phi(P_1)}$, by construction of $\phi(P_2)^{\Rightarrow \phi(P_1)}$, we obtain $v(\mathtt{A}_k) = 0$ for $m \leqslant k \leqslant l$. Therefore $v(\mathtt{A}_m) = 0$ and then, by induction on P_1, we obtain the result.

$P = P_1 \wedge P_2$. This case is done directly by structural induction.

$P = \mathtt{B} \Rightarrow \mathtt{C}\ \mathtt{in}\ P'$ or $P = \mathtt{B} \Rightarrow \mathtt{C}\ \mathtt{in}\ P'$.

If $\mathtt{A} \neq \mathtt{B}$ or $\mathtt{A} \neq \mathtt{C}$ the result is obtained by applying directly structural induction. Let us assume that $\mathtt{A} = \mathtt{B}$ (the case when $\mathtt{A} = \mathtt{C}$ is identical changing $\mathtt{B}$ by $\mathtt{C}$). Let us consider $m = \mathsf{maxin}(\mathtt{B}, \phi(P'))$. If $l \leqslant m$ then $\mathtt{A}_l \in \mathsf{vars}(\phi(P'))$ and we obtain the result by structural induction. So it remains the case when $l = m + 1$. If $l = 0$ there is nothing to prove, so let us assume $l > 0$. Since $v \models \phi(P)$ we obtain $v \models \neg \mathtt{A}_{m+1} \rightarrow \mathtt{A}_m$. Since $v(\mathtt{A}_{m+1}) = 0$ then $m = -1$ or $v(\mathtt{A}_m) = 0$, we us assume $m \geqslant 0$ because if $m = -1$ there is nothing left to prove. Because of the definition of maxin, $\mathtt{A}_m \in \mathsf{vars}(\phi(P'))$. Then, by structural induction, $v(\mathtt{A}_k) = 0$ for $k \leqslant m = l - 1$ that completes the result.

$P = P' \Rightarrow \mathtt{B}$. This case is the same as $P = \mathtt{A}; P$.

$P = P'\backslash \mathtt{B}$. This case is done directly by structural induction. □

Proof (Proposition 9). The proof will be done by structural induction of P. In all cases we are going to use Theorem 1: $\mathtt{prod}(P) = [\![P]\!]$.

$P = \mathtt{nil}$ or $P = \lightning$. These are the base cases, and it is immediate. $\phi(\mathtt{nil})$ is not satisfiable and $\mathtt{prod}(\mathtt{nil}) = \emptyset$. While the valuation v such that $v(\mathtt{A}_k) = 0$ for any $\mathtt{A} \in \mathcal{F}$ and $k \in \mathbb{N}$ meets the thesis for $P = \lightning$.

$P = \mathtt{A};P'$. In this case $\phi(\mathtt{A};P') = \mathtt{A}_{l+1} \wedge \phi(P')$ where $l = \mathrm{maxin}(\mathtt{A},\phi(P'))$.
Let us consider $p \in \mathtt{prod}(P)$, then there is $p' \in \mathtt{prod}(P')$ such that $p = \{\mathtt{A}\} \cup p'$. By applying induction on P' there is a valuation v' that meet the thesis for p' and P'. Let us consider the valuation $v = v'[\mathtt{A}_{l+1}/1]$.[7] It is easy to check that v holds the thesis for p and P:

- $v \models \mathtt{A}_{l+1} \wedge \phi(P')$ since $\mathtt{A}_{l+1} \notin \mathrm{vars}(\phi(P'))$.
- Let us consider $\mathtt{B} \in p$. Then $\mathtt{B} = \mathtt{A}$ or $\mathtt{B} \in p'$. In the first case $v(\mathtt{A}_{l+1}) = 1$. In the second case, by induction, $k \geqslant 0$ and $v'(\mathtt{B}_k) = 1$ where $k = \mathrm{maxin}(\mathtt{B},\phi(P'))$. Then $v'(\mathtt{B}_k) = 1$ and $k = \mathrm{maxin}(\mathtt{B},\phi(P)) = \mathrm{maxin}(\mathtt{B},\phi(P'))$.
- Now let us consider $\mathtt{B} \notin p$. Then $\mathtt{B} \neq \mathtt{A}$ and $\mathtt{B} \notin p'$. By induction, $k = -1$ or $v'(\mathtt{B}_k) = 0$ where $k = \mathrm{maxin}(\mathtt{B},\mathrm{vars}\phi(P'))$. Then $k = \mathrm{maxin}(\mathtt{B},\mathrm{vars}\phi(P))$ and $k < 0$ or $v(\mathtt{B}_k) = 0$.

$P = P_1 \vee P_2$. In this case $\mathtt{prod}(P) = \mathtt{prod}(P_1) \cup \mathtt{prod}(P_2)$. Therefore $p \in \mathtt{prod}(P)$ iff $p \in \mathtt{prod}(P_1)$ or $p \in \mathtt{prod}(P_2)$. Let us suppose $p \in \mathtt{prod}(P_1)$ (the other case is symmetric). By induction there is a valuation v satisfying the thesis for p and P_1. Now let us consider the valuation v' defined in the proof of Lemma 6.2. Let us prove that v' meets the thesis.

- Since $v \models P_1$, then $v' \models \phi(P_1)^{\Rightarrow\phi(P_2)}$.
- Now let us consider a feature $\mathtt{A} \in p$. By induction $v(\mathtt{A}_k) = 1$ where $k = \mathrm{maxin}(\mathtt{A},\phi(P_1))$. Then $v'(\mathtt{A}_k) = 1$ by construction of v'. If $k = \mathrm{maxin}(\mathtt{A},\phi(P))$ we have the result. So, let us suppose $k < \mathrm{maxin}(\mathtt{A},\phi(P))$, let $m = \mathrm{maxin}(\mathtt{A},\phi(P))$. Since $v' \models \phi(P)$, if $v(\mathtt{A}_m) = 0$, by Lemma 7, we could conclude $v(\mathtt{A}_k) = 0$, therefore $v(\mathtt{A}_m) = 1$.
- Finally let us consider $\mathtt{A} \notin p$. By induction $m = -1$ or $v(\mathtt{A}_m) = 0$ where $m = \mathrm{maxin}(\mathtt{A},\phi(P_1))$. If $m = \mathrm{maxin}(\mathtt{A},\phi(P))$, by construction of v', $v'(\mathtt{A}_m) = 0$. If $m < \mathrm{maxin}(\mathtt{A},\phi(P))$, then let us consider $l = \mathrm{maxin}(\mathtt{A},\phi(P_2)) = \mathrm{maxin}(\mathtt{A},\phi(P_1)^{\Rightarrow\phi(P_2)})$. By construction of v',$v'(\mathtt{A}_l) = v(\mathtt{A}_m) = 0$.

$P = \overline{\mathtt{A}};P'$. This case is a particular case of the previous one since $P = {}_E\lightning \vee \mathtt{A};P'$.

$P = P_1 \wedge P_2$. In this case $p \in \mathtt{prod}(P)$ iff there is $p_1 \in \mathtt{prod}(P_1)$ and $p_2 \in \mathtt{prod}(P_2)$ such that $p = p_1 \cup p_2$. By induction on P_1 and P_2, there is a valuation v_1 holding the thesis for p_1 and P_1 and v_2 holding the thesis for p_2 and P_2. Let us consider the valuation v defined as

$$v(x) = \begin{cases} 1 & \text{if } v_1(x) = 1 \text{ and } x \in \mathrm{vars}(\phi(P_1)) \\ 1 & \text{if } v_2(x) = 1 \text{ and } x \in \mathrm{vars}(\phi(P_2)) \\ 0 & \text{othewise} \end{cases}$$

This valuation holds the thesis for p and P:

- Since P is safe and the only operators that introduce negated boolean variables are the restrictions, there are no negated boolean variables in $\phi(P_1)$ or in $\phi(P_2)$. Therefore $v \models \phi(P) = \phi(P_1) \wedge \phi(P_2)$.
- Let us consider a feature $\mathtt{A} \in p$. Then $\mathtt{A} \in p_1$ or $\mathtt{A} \in p_2$. Let us assume $\mathtt{A} \in p_1$, the other case is symmetric. Then, by induction, $v_1(\mathtt{A}_l) = 1$ where $l = \mathrm{maxin}(\mathtt{A},\phi(P_1))$. Then, by construction of v, $v(\mathtt{A}_l) = 1$ Let us consider $m = \mathrm{maxin}(\mathtt{A},\phi(P))$. If $v(\mathtt{A}_m) = 0$, by Lemma 7 $v(\mathtt{A}_l) = 0$, so $v(\mathtt{A}_m) = 1$
- Let us consider $\mathtt{A} \notin p$. Then $\mathtt{A} \notin p_1$ and $\mathtt{A} \notin p_1$ Let us consider $l = \mathrm{maxin}(\mathtt{A},\phi(P))$. then $l = \mathrm{maxin}(\mathtt{A},\phi(P_1))$ or $l = \mathrm{maxin}(\mathtt{A},\phi(P_2))$. Let us suppose $l = \mathrm{maxin}(\mathtt{A},\phi(P_1))$, the other case is symmetric. Since $\mathtt{A} \notin p_1$, $l = -1$ or $v_1(\mathtt{A}) = 0$. There two possibilities $l = \mathrm{maxin}(\mathtt{A},\phi(P_2))$ or $l < \mathrm{maxin}(\mathtt{A},\phi(P_2))$. In the second case $\mathtt{A}_l \notin vars(\phi(P_2))$ so, by construction of v,$v(\mathtt{A}_l) = 0$. In the first case, by induction, $l = -1$ or $v_2(\mathtt{A}_l) = 0$; so, by construction of v,$v(\mathtt{A}_l) = 0$.

$P = \mathtt{A} \Rightarrow \mathtt{B}\,\mathrm{in}\,P'$. In this case $\phi(P) = (\neg\mathtt{A}_{l+1} \to \neg\mathtt{A}_l) \wedge (\neg\mathtt{B}_{m+1} \to \neg\mathtt{B}_m) \wedge (\mathtt{A}_{l+1} \to \mathtt{B}_{m+1}) \wedge \phi(P')$ where $l = \mathrm{maxin}(\mathtt{A},\phi(P'))$ and $m = \mathrm{maxin}(\mathtt{B},\phi(P'))$.
Let us consider $p \in \mathtt{prod}(P)$ iff there is $p' \in \mathtt{prod}(P')$ such that $p = p'$ and $\mathtt{A} \notin p'$ or $p = p' \cup \{\mathtt{B}\}$ and $\mathtt{A} \in p'$. In both cases, by induction, there is a valuation v' satisfying the thesis for p' and for P'.

Let us consider the valuation v according to the following cases:

$\mathtt{A} \in p'$. Then $v = v'[\mathtt{A}_{l+1}/1, \mathtt{B}_{m+1}/1]$.
$\mathtt{A} \notin p'$ and $\mathtt{B} \in p'$. Then $v = v'[A_{l+1}/0,B_{m+1}/1]$.
$\mathtt{A} \notin p'$ and $\mathtt{B} \notin p'$. Then $v = v'[A_{l+1}/0,B_{m+1}/0]$.

It is easy to check that all three cases v satisfy the conditions for p and P:

- Due to the way v and $\phi(P)$ are defined, $v \models \phi(P)$.
- Let us consider $\mathtt{C} \in p$. If $\mathtt{C} = \mathtt{B}$ there are two cases $\mathtt{B} \in p'$ or $\mathtt{A} \in p'$; in both cases, by construction of v,$v(\mathtt{C}_{m+1}) = 1$. If $\mathtt{C} = \mathtt{A}$ then $\mathtt{A} \in p'$, so $v(\mathtt{A}_{l+1}) = 1$. If $\mathtt{C} \neq \mathtt{B}$ and $\mathtt{C} \neq \mathtt{B}$, then $\mathtt{C} \in p'$ and by induction $l \geqslant 0$ and $v'(\mathtt{C}_l) = v(\mathtt{C}_l) = 1$ for $l = \mathrm{maxin}(\mathtt{C},\phi(P')) = \mathrm{maxin}(\mathtt{C},\phi(P))$
- Let us consider $\mathtt{C} \notin p$. Then $\mathtt{C} \notin p'$. By induction $l < 0$ or $v'(\mathtt{C}_l) = 0$ for $l = \mathrm{maxin}(\mathtt{C},\phi(P'))$. If $\mathtt{C} \neq \mathtt{A}$ or $\mathtt{C} \neq \mathtt{A}$, then $l = \mathrm{maxin}(\mathtt{C},\phi(P'))$ and $l < 0$ or $v(\mathtt{C}_l) = v'(\mathtt{C}_l) = 0$. If $\mathtt{C} = \mathtt{B}$, then $\mathtt{B} \notin p'$ and, by construction of v,$v(\mathtt{B}_{m+1}) = 0$. If $\mathtt{C} = \mathtt{A}$, then $\mathtt{A} \notin p'$ and, by construction of v, $v(\mathtt{A}_{l+1}) = 0$

$P = \mathtt{A} \nRightarrow \mathtt{B}\,\mathrm{in}\,P'$. In this case $\phi(P) = (\neg\mathtt{A}_{l+1} \to \neg\mathtt{A}_l) \wedge (\neg\mathtt{B}_{m+1} \to \neg\mathtt{B}_m) \wedge (\neg\mathtt{A}_{l+1} \vee \neg\mathtt{B}_{m+1}) \wedge \phi(P')$ where $l = \mathrm{maxin}(\mathtt{A},\phi(P'))$ and $m = \mathrm{maxin}(\mathtt{B},\phi(P))$. Moreover, $p \in \mathtt{prod}(P)$ iff $p \in \mathtt{prod}(P')$ and $\mathtt{A} \notin p$ or $\mathtt{B} \notin p$.
Let us assume the first case (the other is symmetric). By induction hypothesis there is a valuation v' satisfying the thesis for p and P'. Now there are two cases $\mathtt{B} \in p$ or $\mathtt{B} \notin p$, in the first case let us consider $v = v'[\mathtt{A}_{l+1}/0,\mathtt{B}_{m+1}/1]$, in the second case let us consider $v = v'[\mathtt{A}_{l+1}/0,\mathtt{B}_{m+1}/0]$. Let us check that v satisfies the thesis for p and P:

- Let us check that $v \models \phi(P)$ by analyzing its parts. Since $\mathtt{A} \notin p$, by induction, $l < 0$ or $v(\mathtt{A}_l) = 0$, so $v \models \neg\mathtt{A}_{l+1} \to \neg\mathtt{A}_l$. Also by induction, $\mathtt{B} \in p$ iff $\mathtt{B} \in p'$ iff $m \geqslant 0$ and $v'(\mathtt{B}_m) = 1$. Therefore $v \models \neg\mathtt{B}_{m+1} \to \neg\mathtt{B}_m$. By construction $v \models \neg\mathtt{A}_{l+1} \vee \neg\mathtt{B}_{m+1}$ and by induction $v' \models \phi(P')$. Since $\mathtt{A}_{l+1},\mathtt{B}_{m+1} \notin \mathrm{vars}(\phi(P'))$,$v \models \phi(P')$.
- Let us consider $\mathtt{C} \in p$, since we are considering $\mathtt{A} \notin p'$,$\mathtt{C} \neq \mathtt{A}$. If $\mathtt{C} = \mathtt{B}$, then $\mathtt{B} \in p'$. Therefore, by construction of v, $v(\mathtt{B}_{m+1}) = 1$.

[7] $v[A/x](B) = v(B)$ if $A \neq B$ and $v[A/x](A) = x$.

Otherwise by induction hypothesis, $k \geqslant 0$ and $v'(\mathtt{C}_k) = 1$ for $k = \mathsf{maxin}(\mathtt{C},\phi(P'))$. Then $k = \mathsf{maxin}(\mathtt{C},\phi(P))$ and $v(\mathtt{C}_k) = v'(\mathtt{C}_k) = 1$.

- Let us consider $\mathtt{C} \notin p$. If $\mathtt{C} = \mathtt{A}$ then $v(\mathtt{A}_{l+1}) = 0$. If $\mathtt{C} = \mathtt{B}$ then $v(\mathtt{B}_{m+1}) = 0$. If $\mathtt{C} \neq \mathtt{A}$ and $\mathtt{C} \neq \mathtt{B}$, by induction $k < 0$ or $v'(\mathtt{C}_k) = 0$ for $k = \mathsf{maxin}(\mathtt{C},\phi(P'))$. Let us observe that $k = \mathsf{maxin}(\mathtt{C},\phi(P))$ and $v(\mathtt{C}_k) = v'(\mathtt{C}_k) = 0$

$P = P' \Rightarrow \mathtt{A}$. This case is the same as $P = \mathtt{A};P$.

$P = P'\backslash\mathtt{A}$. In this case $\phi(P) = \neg\, \mathtt{A}_l \wedge \phi(P)$ where $l = \mathsf{maxin}(\mathtt{A},\phi(P'))$. Let us consider $p \in \mathtt{prod}(P)$, then there is $p \in \mathtt{prod}(P')$ such that $\mathtt{A} \notin p$. By structural induction there is a valuation v holding the thesis for p and P'. Let us check that v also satisfies the conditions for P.

- First $v \models \phi(P)$. There are two cases: $l = -1$ or $l \geqslant 0$. In the first case, the $\mathtt{A}_{l-1}$ is the symbol $\perp$. In the second case, since $\mathtt{A} \notin p$, by induction $v(\mathtt{A}_l) = 0$. In any case, by induction hypothesis, $v \models \neg\mathtt{A}_l \wedge \phi(P')$.
- Let us consider a feature $\mathtt{B}$. By induction, $\mathtt{B} \in p$ iff $l > 0$ and $v(\mathtt{B}_l) = 1$ where $l = \mathsf{maxin}(\mathtt{B},\phi(P'))$. To conclude this item it is enough to consider that $\mathsf{vars}(\phi(P)) = \mathsf{vars}(\phi(P'))$.

The second condition holds trivially since $\mathsf{vars}(\phi(P)) = \mathsf{vars}(\phi(P'))$. □

Proof Proposition 10. The proof will be done by structural induction on P.

$P = \mathtt{nil}$ or $P = \lightning$. These are the base cases, and it is immediate. $\phi(\mathtt{nil})$ is not satisfiable and $\mathtt{prod}(\mathtt{nil}) = \emptyset$. While the only product of $\lightning$, the empty set $\emptyset$, has no features.

$P = \mathtt{A};P'$. In this case $\phi(\mathtt{A};P') = \mathtt{A}_{l+1} \wedge \phi(P')$ where $l = \max(0,\mathsf{maxin}(\mathtt{A},\phi(P')))$. If $v \models \phi(P)$, then $v \models \phi(P')$. By structural induction there is $p' \in \mathtt{prod}(P')$, satisfying the thesis. Let us consider $p = p' \cup \{\mathtt{A}\} \in \mathtt{prod}(P)$. By the definition of the denotational semantics of the prefix operator, $p \in \mathtt{prod}(\mathtt{A};P')$.
Now let us consider a feature $\mathtt{C}$ such that $v(\mathtt{C}_k) = 0$ for all $0 \leqslant k \leqslant \mathsf{maxin}(\mathtt{C},\phi(P))$. Since $v \models \phi(P)$, $v(A_{l+1}) = 0$,$\mathtt{C} \neq \mathtt{A}$. By induction, $\mathtt{C} \notin p'$, then, by construction of p,$\mathtt{C} \notin p$.

$P = P_1 \vee P_2$. In this case $\phi P = \phi(P_1)^{\Rightarrow\phi(P_2)} \vee \phi(P_2)^{\Rightarrow\phi(P_1)}$. Since $v \models \phi(P)$, then $v \models \phi(P_1)^{\Rightarrow\phi(P_2)}$ or $v \models \phi(P_2)^{\Rightarrow\phi(P_1)}$. Let us suppose $v \models \phi(P_2)^{\Rightarrow\phi(P_1)}$ (the other case is symmetrical). By Lemma 6.1, $v \models \phi P_2$. Then, by induction, there is a product $p \in \mathtt{prod}(P_2)$ that meet the thesis for p and P_2. Since $\mathtt{prod}(P) = \mathtt{prod}(P_1) \cup \mathtt{prod}(P_2)$, $p \in \mathtt{prod}(P)$. Now let us consider a feature $\mathtt{A}$ such that $v(\mathtt{A}_l) = 0$ for all $0 \leqslant l \leqslant \mathsf{maxin}(\mathtt{A},\phi(P))$. Since $\mathsf{maxin}(\mathtt{A},\phi(P)) \geqslant \mathsf{maxin}(\mathtt{A},\phi(P_2))$, by induction, $\mathtt{A} \notin p$.

$P = \overline{\mathtt{A}};P'$. This case is a particular case of the previous one since $P = {}_E\lightning \vee \mathtt{A};P'$.

$P = P_1 \wedge P_2$. In this case $\phi(P) = \phi(P_1) \wedge \phi(P_2)$. Since $v \models \phi(P)$ then $v \models \phi(P_1)$ and $v \models \phi(P_2)$. By structural induction there is $p_1 \in \mathtt{prod}(P_1)$ that meet the thesis for v and P_1 and there is $p_2 \in \mathtt{prod}(P_2)$ that meet the thesis for v and P_2. Let us show $p = p_1 \cup p_2$ meet the thesis for v and P. First, by the definition of the operators $p \in \mathtt{prod}(P)$. Now let us consider a feature $\mathtt{A}$ such that $v(\mathtt{A}_l) = l$ for all $0 \leqslant k \leqslant \mathsf{maxin}(\mathtt{A},\phi(P))$. Since $\mathsf{maxin}(\mathtt{A},\phi(P)) \geqslant \mathsf{maxin}(\mathtt{A},\phi(P_1))$ and $\mathsf{maxin}(\mathtt{A},\phi(P)) \geqslant \mathsf{maxin}(\mathtt{A},\phi(P_2))$, by induction $\mathtt{A} \notin p_1$ and $\mathtt{A} \notin p_2$. Therefore $\mathtt{A} \notin p_1 \cup p_2 = p$.

$P = \mathtt{A} \Rightarrow \mathtt{B}\,\mathtt{in}\,P'$. In this case $\phi(P) = (\neg\mathtt{A}_{l+1} \to \neg\mathtt{A}_l) \wedge (\neg\mathtt{B}_{m+1} \to \neg\mathtt{B}_m) \wedge (\mathtt{A}_{l+1} \to \mathtt{B}_{m+1}) \wedge \phi(P')$ where $l = \mathsf{maxin}(\mathtt{A},\phi(P'))$ and $m = \mathsf{maxin}(\mathtt{B},\phi(P'))$. Let us consider v a valuation such that $v \models \phi(P)$. By induction, there is $p' \in \mathtt{prod}(P')$ holding the thesis for v and P.
There are two cases $\mathtt{A} \in p'$ or $\mathtt{A} \notin p'$. In the first case let us consider $p = p' \cup \{\mathtt{B}\} \in \mathtt{prod}(P)$, and in the second case let $p = p'$. Let us check that, in both cases, p hold the thesis for v and P. By the definition of the denotational semantics, $p \in \mathtt{prod}(P)$. Now let us consider a feature $\mathtt{C}$ such that $v(\mathtt{C}_k) = 0$ for all with $k \in \mathbb{N}$. There are the following cases:

$\mathtt{C} = \mathtt{A}$. By induction $\mathtt{A} \notin p'$. Then, by construction of p, $\mathtt{A} \notin p$.

$\mathtt{C} = \mathtt{B}$. By induction $\mathtt{B} \notin p'$. Since $v(\mathtt{B}_{m+1}) = 0$ and $v \models \phi(P)$,$v(\mathtt{A}_{l+1}) = 0$. Then, $l = -1$ or $v(\mathtt{A}_l) = 0$. Then, by Lemma 7, $v(\mathtt{A}_k) = 0$ for all $0 \leqslant k < l$. So, by induction, $\mathtt{A} \notin p'$. Since $\mathtt{B} \notin p'$ and $\mathtt{A} \notin p'$, by construction of p, $\mathtt{B} \notin p$.

$\mathtt{C} \neq \mathtt{B}$ and $\mathtt{C} \neq \mathtt{A}$. In this case, by induction, $\mathtt{C} \notin p'$. Since $\mathtt{C} \neq \mathtt{B}$ and $\mathtt{C} \neq \mathtt{A}$, by construction of p,$\mathtt{C} \notin p$.

$P = \mathtt{A} \not\Rightarrow \mathtt{B}\,\mathtt{in}\,P'$. In this case $\phi(P) = (\neg\mathtt{A}_{l+1} \to \neg\mathtt{A}_l) \wedge (\neg\mathtt{B}_{m+1} \to \neg\mathtt{B}_m) \wedge (\neg\mathtt{A}_{l+1} \vee \neg\mathtt{B}_{m+1}) \wedge \phi(P')$ where $l = \mathsf{maxin}(\mathtt{A},\phi(P'))$ and $m = \mathsf{maxin}(\mathtt{B},\phi(P'))$.
Let us consider v such that $v \models \phi(P)$. Then $v \models \phi(P')$, and by induction, there is $p \in \mathtt{prod}(P')$ that holds the thesis for v and P'. Let us show that p also holds the thesis for v and P. First we will prove that $p \in \mathtt{prod}(P)$. Since $v \models \phi(P)$, $v(\mathtt{A}_{l+1}) = 0$ or $v(\mathtt{B}_{m+1}) = 0$; let us assume $v(\mathtt{B}_{m+1}) = 0$, the other case is symmetric. Since $v \models \phi(P)$, $m = -1$ or $v(\mathtt{B}_m) = 0$, then, by Lemma 7, $v(\mathtt{B}_k) = 0$ for $0 \leqslant k \leqslant m$. Then, by induction, $\mathtt{B} \notin p$. So, by the definition of the denotational semantics, $p \in \mathtt{prod}(P)$.
Now let us consider a feature $\mathtt{C}$ such that $v(\mathtt{C}_k) = 0$ for all $0 \leqslant k \leqslant \mathsf{maxin}(\mathtt{C},\phi(P))$. There are the following cases:

$\mathtt{C} = \mathtt{A}$. Then $k = l + 1$ and $l = \mathsf{maxin}(\mathtt{A},\phi(P'))$. Since $v \models \phi(P)$ and $v(\mathtt{A}_{l+1}) = 0$,$l = -1$ or $v(\mathtt{A}_l) = 0$. Then, by Lemma 7, $v(\mathtt{A}_k) = 0$ for $0 \leqslant k \leqslant l$. Therefore, by induction, $\mathtt{A} \notin p$.

$\mathtt{C} = \mathtt{B}$. Since $v \models \phi(P)$ and $v(\mathtt{B}_{m+1}) = 0$,$m = -1$ or $v(\mathtt{B}_m) = 0$. Then by Lemma 7, $v(\mathtt{B}_k) = 0$ for $0 \leqslant k \leqslant m$. Then, by induction $\mathtt{B} \notin p$.

$\mathtt{C} \neq \mathtt{B}$ and $\mathtt{C} \neq \mathtt{A}$. In this case it is enough to consider that $\mathsf{maxin}(\mathtt{C},\phi(P')) = \mathsf{maxin}(\mathtt{C},\phi(P))$. Then, by induction, $\mathtt{C} \notin p$.

$P = P' \Rightarrow \mathtt{A}$. This case is the same as $P = \mathtt{A};P$.

$P = P'\backslash\mathtt{A}$. In this case $\phi(P) = \neg\mathtt{A}_l \wedge \phi(P')$ where $l = \mathsf{maxin}(\mathtt{A},\phi(P'))$. Let us consider a valuation v such that $v \models \phi(P)$. Then $v \models \phi(P')$. By induction there is $p \in \mathtt{prod}(P')$ holding the thesis for v and P'. Let us check that p holds the thesis for v and P. First, let us show that $\mathtt{A} \notin p$. Since $v \models \phi(P)$, there are two cases $l = -1$ or $l \geqslant 0$ and $v(\mathtt{A}_l) = 0$. Then, by Lemma 7, $v(\mathtt{A}_k) = 0$ for $0 \leqslant k \leqslant l$.

Then, by induction, $\mathtt{A} \notin p$. So, by the definition of the semantic operator $[\![\cdot \backslash \mathtt{A}]\!], p \in \mathtt{prod}(P)$. Now let us consider a feature $\mathtt{C}$ such that $v(\mathtt{C}_k) = 0$ for $0 \leqslant k \leqslant \mathsf{maxin}(\mathtt{C}, \phi(P))$. We have already proved that $\mathtt{A} \notin p$, so we can assume that $\mathtt{C} \neq \mathtt{A}$. Since $\mathsf{maxin}(\mathtt{C}, \phi(P)) = \mathsf{maxin}(\mathtt{C}, \phi(P'))$, we obtain $\mathtt{C} \notin P$ by induction. □

References

[1] P. Clements, L. Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley, 2001.

[2] K. Pohl, G. Böckle, F. van der Linden, Software Product Line Engineering: Foundations, Principles and Techniques, Springer, 2005.

[3] S. Kotha, From mass production to mass customization: the case of the national industrial bicycle company of Japan, European Management Journal 14 (5) (1996) 442–450, http://dx.doi.org/10.1016/0263-2373(96)00037-0.

[4] R. Milner, A Calculus of Communicating Systems (LNCS 92), Springer, 1980.

[5] C. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.

[6] M. Hennessy, Algebraic Theory of Processes, MIT Press, 1988.

[7] R. Hierons, J. Bowen, M. Harman (Eds.), Formal Methods and Testing, LNCS 4949, Springer, 2008. doi:http://dx.doi.org/10.1007/978-3-540-78917-8.

[8] R. Hierons, K. Bogdanov, J. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, A. Simons, S. Vilkomir, M. Woodward, H. Zedan, Using formal specifications to support testing, ACM Computing Surveys 41 (2) (2009) 9:1–9:76, http://dx.doi.org/10.1145/1459352.1459354.

[9] I. Rodríguez, A general testability theory, 20th International Conference on Concurrency Theory, CONCUR'09, LNCS 5710, Springer, 2009, pp. 572–586. doi:http://dx.doi.org/10.1007/978-3-642-04081-8_38.

[10] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, Feature-Oriented Domain Analysis (FODA) feasibility study, Tech. Rep. CMU/SEI-90-TR-21, Carnegie Mellon University, 1990.

[11] J. Bosch, Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach, Addison-Wesley, 2000.

[12] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, J. Zettel, Component-Based Product Line Engineering with UML, Addison-Wesley, 2002.

[13] C. Krzysztof, H. Simon, W. Ulrich, Staged configuration through specialization and multilevel configuration of feature models, Software Process: Improvement and Practice 10 (2) (2005) 143–169.

[14] D. Fischbein, S. Uchitel, V. Braberman, A foundation for behavioural conformance in software product line architectures, in: Workshop on Role of Software Architecture for Testing and Analysis, ROSATEA'06, ACM Press, 2006, pp. 39–48. doi:http://dx.doi.org/10.1145/1147249.1147254.

[15] A. Fantechi, S. Gnesi, A behavioural model for product families, in: 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers, ESEC-FSE companion '07, ACM Press, 2007, pp. 521–524. doi:http://dx.doi.org/10.1145/1287624.1287700.

[16] K. Larsen, U. Nyman, A. Wasowski, Modal I/O automata for interface and product line theories, in: 16th European Conference on Programming, ESOP'07, Springer, 2007, pp. 64–79. doi:http://dx.doi.org/10.1007/978-3-540-71316-6_6.

[17] K.G. Larsen, U. Nyman, A. Wasowski, On modal refinement and consistency, in: 18th International Conference on Concurrency Theory, CONCUR'07, LNCS 4703, 2007, pp. 105–119. doi:http://dx.doi.org/10.1007/978-3-540-74407-8_8.

[18] A. Fantechi, S. Gnesi, Formal modeling for product families engineering, in: 12th International Software Product Line Conference, SPLC'08, IEEE Computer Society Press, 2008, pp. 193–202. doi:http://dx.doi.org/10.1109/SPLC.2008.45.

[19] P. Asirelli, M.H. ter Beek, A. Fantechi, S. Gnesi, A logical framework to deal with variability, in: 8th International Conference on Integrated Formal Methods, IFM'10, Springer, 2010, pp. 43–58. doi:http://dx.doi.org/10.1007/978-3-642-16265-7_5.

[20] P. Asirelli, M.H. ter Beek, A. Fantechi, S. Gnesi, F. Mazzanti, Design and validation of variability in product lines, in: 2nd International Workshop on Product Line Approaches in Software Engineering, PLEASE '11, ACM, 2011, pp. 25–30. doi:http://dx.doi.org/10.1145/1985484.1985492.

[21] P. Asirelli, M.H. ter Beek, S. Gnesi, A. Fantechi, Formal description of variability in product families, in: 15th International Software Product Line Conference, SPLC '11, IEEE Computer Society Press, 2011, pp. 130–139. doi:http://dx.doi.org/10.1109/SPLC.2011.34.

[22] D. Batory, Feature models, grammars, and propositional formulas, in: 9th International Software Product Line Conference, SPLC'05, Springer, 2005, pp. 7–20. doi:http://dx.doi.org/10.1007/11554844_3.

[23] D. Benavides, S. Segura, A. Ruiz, Automated analysis of feature models 20 years later: a literature review, Information Systems 35 (6) (2010) 615–636, http://dx.doi.org/10.1016/j.is.2010.01.001.

[24] P.Y. Schobbens, P. Heymans, J.-C. Trigaux, Y. Bontemps, Generic semantics of feature diagrams, Computer Networks 51 (2) (2007) 456–479, http://dx.doi.org/10.1016/j.comnet.2006.08.008.

[25] R. Muschevici, J. Proença, D. Clarke, Modular modelling of software product lines with feature nets, in: 9th International Conference Software Engineering and Formal Methods, SEFM'11, 2011, pp. 318–333.

[26] P. Höfner, R. Khédri, B. Möller, Feature algebra, 14th International Symposium on Formal Methods, FM'06, LNCS 4085, Springer, 2006, pp. 300–315.

[27] P. Höfner, R. Khédri, B. Möller, An algebra of product families, Software and System Modeling 10 (2) (2011) 161–182, http://dx.doi.org/10.1007/s10270-009-0127-2.

[28] A. Gruler, M. Leucker, K. Scheidemann, Modeling and model checking software product lines, 10th IFIP WG 6.1 International Conference, FMOODS'08, LNCS 5051, Springer, 2008, pp. 113–131.

[29] M. Mannion, Using first-order logic for product line model validation, in: 2nd International Software Product Line Conference, SPLC'02, Springer, 2002, pp. 176–187.

[30] Y. Bontemps, P. Heymans, P. Schobbens, J. Trigaux, Semantics of FODA feature diagrams, in: 1st Workshop on Software Variability Management for Product Derivation Towards Tool Support, SPLCW'04, Springer, 2004, pp. 48–58.

[31] K. Czarnecki, S. Helsen, Feature-based survey of model transformation approaches, IBM Systems Journal 45 (3) (2006) 621–646.

[32] P. Heymans, P. Schobbens, J. Trigaux, Y. Bontemps, R. Matulevicius, A. Classen, Evaluating formal properties of feature diagram languages, IET Software 2 (3) (2008) 281–302.

[33] M. Mendonça, A. Wasowski, K. Czarnecki, SAT-based analysis of feature models is easy, in: 13rd International Software Product Line Conference, SPLC'09, 2009, pp. 231–240, doi:http://dx.doi.org/10.1145/1753235.1753267.

[34] P. Chen, The entity-relationship model toward a unified view of data, ACM Transactions on Database Systems 1 (1976) 9–36.

[35] M. Griss, J. Favaro, Integrating feature modeling with the RSEB, in: 5th Interantional Conference on Software Reuse, ICSR'98, 1998, pp. 76–85.

[36] M. Eriksson, J. Borstler, K. Borg, The pluss approach – domain modeling with features, use cases and use case realizations, in: 9th International Conference on Software Product Lines, SPLC'06, Springer-Verlag, 2006, pp. 33–44.

[37] H. Palikareva, J. Ouaknine, A.W. Roscoe, SAT-solving in CSP trace refinement, Science Computer Programming 77 (10–11) (2012) 1178–1197. http://dx.doi.org/10.1016/j.scico.2011.07.008.

[38] H. Palikareva, J. Ouaknine, B. Roscoe, Faster fdr counterexample generation using sat-solving, ECEASST 23.

[39] Y. Liu, J. Sun, J.S. Dong, An analyzer for extended compositional process algebras, in: Companion of the 30th International Conference on Software Engineering, ICSE Companion '08, ACM, New York, NY, USA, 2008, pp. 919–920. doi:http://dx.doi.org/10.1145/1370175.1370187.

[40] S. Segura, R. Hierons, D. Benavides, A. Ruiz, Automated metamorphic testing on the analyses of feature models, Information & Software Technology 53 (3) (2011) 245–258, http://dx.doi.org/10.1016/j.infsof.2010.11.002.

[41] A. Classen, Q. Boucher, P. Heymans, A text-based approach to feature modelling: syntax and semantics of TVL, Science of Computer Programming 76 (12) (2011) 1130–1143, http://dx.doi.org/10.1016/j.scico.2010.10.005.

[42] L. Llana, M. Núñez, I. Rodríguez, Derivation of a suitable finite test suite for customized probabilistic systems, Formal Techniques for Networked and Distributed Systems – FORTE 2006, LNCS 4229, Springer, 2006, pp. 467–483. doi:http://dx.doi.org/10.1007/11888116_33.