IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 25, 2022, accepted March 17, 2022, date of publication April 6, 2022, date of current version April 14, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3165185

A Finite Prefix for Analyzing Information Flow Among Transitions of a Free-Choice Net

FEDERICA ADOBBATI[1], GÖRKEM KILINÇ SOYLU[2], AND ADRIÁN PUERTO AUBEL[3]

[1]Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, 20126 Milano, Italy
[2]Department of Computer Science, Technical University of Darmstadt, 64289 Darmstadt, Germany
[3]Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, 9747 AG Groningen, The Netherlands

Corresponding author: Görkem Kılınç Soylu (gorkemklnc@gmail.com)

The work of Federica Adobbati was supported by the Italian Ministero dell'Università e della Ricerca. Görkem Kılınç Soylu gratefully acknowledges support by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cyber-security ATHENE. The work of Adrián Puerto Aubel was by the Austrian Science Fund FWF under Project P33548.

ABSTRACT In distributed systems, the occurrence of an action can give information about the occurrence of other actions. This can be an unwanted situation when ''high'' actions of the system need to be kept secret, while allowing users to observe ''low'' actions. If it is possible to deduce information about occurrence of high actions by observing only low actions, then the system suffers from an unwanted information flow. ''Reveals'' and ''excludes'' relations were introduced for modelling and analysing such an information flow among actions of a distributed system that is modelled via Petri nets. In this paper, we provide a formal basis for computing reveals and excludes relations of 1-safe free-choice Petri nets. We introduce the ''maximal-step computation tree'' to represent the behaviour of a distributed system under maximal-step semantics. We define a finite prefix of the tree called ''full prefix'' and we show that it is adequate for analysing information flow by means of reveals and excludes relations.

INDEX TERMS Concurrency, distributed systems, excludes relation, finite prefix, free-choice nets, full prefix, information flow, maximal-step computation tree, Petri nets, reveals relation.

I. INTRODUCTION AND RELATED WORK

In this paper, we work on a formal basis for modelling and analysing information flow in distributed systems. We consider systems that are modelled with 1-safe free-choice Petri nets.

Petri nets are a formalism for modelling concurrent and distributed systems. The concept of Petri nets originates from the PhD thesis of Carl Adam Petri [1] written in the early sixties. Since then, Petri nets have been extensively studied, developed and applied in many different areas. A comprehensive overview on the Petri net theory and its applications is given in [2], [3] and [4]. A more recent overview on Petri nets is presented in [5] from a systems theory and automatic control perspective. Decidability results addressing important properties and equivalence notions of Petri nets are collected in [6].

A Petri net is given by a set of *places*, and by a set of *transitions*. Places may contain *tokens*. The state of a Petri net is given by the distribution of tokens among the whole set of places. When a transition occurs, or *fires*, it may alter the distribution of tokens among some places. In this way, Petri nets can explicitly encode relations of conflict, causal dependence and concurrency between transitions.

In the literature, a variety of Petri net classes are introduced. These classes differ in the nature of their places. In the basic classes (elementary net systems, 1-safe Petri nets), places represent logical conditions, which can be true or false. The state of the net (or the *marking*) is then subset of places which hold the value true. In *Place/Transition nets*, places are counters, which can take nonnegative integer values. In high-level nets, a place is a container for *tokens* whose values range over more general types. Further extensions of the basic model introduce probabilistic or temporal parameters.

Formally, the structure of a basic Petri net is given by the *flow relation*, which links places and transitions. Restricted classes of Petri nets are defined by imposing restrictions on that structure. Here, we are interested in so-called free-choice nets.

The associate editor coordinating the review of this manuscript and approving it for publication was S. K. Hafizul Islam.

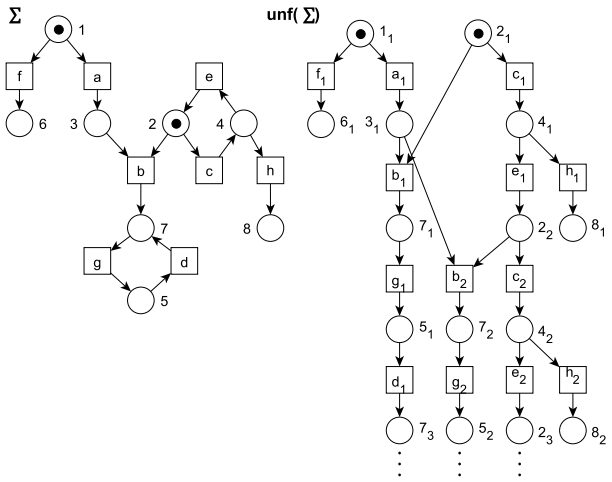


FIGURE 1. A 1-safe Petri net and its unfolding.

The class of free-choice Petri nets, initially studied in [7], is relevant in Petri net theory since a large amount of results relates the structure of the net to its behaviour, yielding efficient analysis algorithms. See [8] and [9] for a collection of the main results on this class.

Free-choice Petri nets allow for modelling systems whose behaviour include both conflicts and synchronisations as well as concurrency, but they rule out situations in which conflicts and synchronisations interfere with each other, the so called situations of confusion. A typical example of confusion (asymmetric confusion) is given in the net of Fig. 1: transitions a and c are concurrently enabled, whereas b is not enabled. Due to the synchronisation at b, the occurrence of a influences which one between b and c can occur. If a occurs before c there will be a choice between b and c; if a occurs after c, then b will still not be enabled.

Since confusion is excluded, this class of nets cannot model situation of mutual exclusion or resource sharing as discussed in [10]; in spite of that, free-choice nets are suitable to model the flow of control in networks of processors and they are widely used in many application domains as for example in process mining [11], [12]. In [13] and [14], Van der Aalst provides new results in the theory of free-choice nets, including reduction methods to systematically prove properties, thus improving the efficiency of analysis.

The behaviour of a Petri net can be described in different ways. The most used models are the marking graph and the unfolding [15]. The marking graph is a labelled transitions system, formed by the reachable markings, connected by arcs corresponding to transition firings. The unfolding of a Petri net is an acyclic Petri net, made of occurrences of the elements of the original net, both transitions and places, partially ordered by a ''causality'' relation. In the early stages of this research project, we explored the use of unfolding and unfolding prefixes, namely the complete prefix [16] and the canonical prefix [17]. However, we saw that in the case of free-choice nets under the maximal-step semantics, representation of the behaviour as a tree structure allows us to give simpler axiomatic definitions and provide algorithms.

In the standard semantics, transitions fire asynchronously; an alternative is given by step semantics, where actions that are concurrently enabled occur simultaneously. Maximal-step semantics, in particular, requires that as many actions as possible occur at a time. An overview of different semantics can be found in [4]. The relation between step semantics and maximal-step semantics has been studied for example in [18], where it is shown that they are equivalent in the case of systems without asymmetric confusion, and therefore also in the case of free-choice Petri nets. In our setting, it is assumed that the system progresses unless it goes into a deadlock. Also, the order of occurrence of concurrently enabled actions, or the states reached during occurrence of these concurrent actions have no significance to our purpose. In fact, we can assume the concurrent actions that are enabled at the same state can occur simultaneously.

In distributed systems, security considerations may require that a user is not able to infer if a given action has been performed by another component, while still being able to interact with that component. This kind of unwanted information flow has been studied in formal models of concurrent systems through the concept of noninterference. An informative brief overview on information flow and noninterference is provided in [19].

The concept of noninterference was introduced by Goguen and Meseguer for deterministic state machines in [20]. Sutherland and McCullough moved the concept to the nondeterministic and concurrent systems in [21] and [22]. Since then, various noninterference properties have been proposed in the literature based on different system models. Focardi and Gorrieri attempt to provide a classification of possibilistic security properties for process algebras. They do not, however, provide a general scheme for the definition of security [23]. By the use of selective interleaving functions, McLean provides a uniform framework in [24] which allows one to compare a subset of different information-flow security properties and reason about composition. Mantel provides a more expressive framework in which different noninterference notions can be defined in a modular way by assembling basic security predicates [25], [26]. As shown by Mantel in [27], each property that can be represented in McLean's framework of selective interleaving functions can also be represented in the assembly kit.

In the concept of noninterference, a system is viewed as consisting of components at two distinct levels of confidentiality: *high* (hidden) and *low* (observable). A system is then said to be secure with respect to noninterference if a user, which knows the structure of the system, cannot deduce information about high actions by interacting only via low actions. Busi and Gorrieri moved the concept to 1-safe Petri nets in [28] by studying observational equivalences and structural properties. In [29], the authors define structural noninterference properties for elementary nets

based on absence of particular places in the net. Frau *et al.*, in [30], investigate the algorithmic properties of these structural properties and introduce a tool for detecting places that violate security in elementary net models. Best *et al.*, in [31], study the decidability of noninterference in Petri nets and prove that it is undecidable for unbounded nets. Baldan and Carraro give a characterisation of noninterference based on unfoldings of 1-safe Petri nets in terms of causalities and conflicts in [32]. In [33], multilevel noninterference properties are studied based on causal characterisations in the unfolding semantics of safe net systems. In [34], the authors provide an algorithm to compute all the minimal solutions for enforcing noninterference on bounded Petri nets by using linear integer programming techniques. In this work, we follow the line of results from the references [35] and [36] which introduce two families of relations, namely *reveals* and *excludes*, to model information flow among transitions of a Petri net and use these relations to define various noninterference properties for 1-safe Petri nets. The original reveals relation was defined for events of an occurrence net by Haar in [37]. In [38] it was applied in the field of fault diagnosis. Later, in [36], the relation was redefined for transitions of a Petri net and applied in the field of information-flow security by means of noninterference.

Noninterference deals with two kinds of information flow: positive and negative. The first one arises when the occurrence of a transition allows an observer to infer that another transition has already occurred or will inevitably occur in the future. The second one refers to the deduction of information about nonoccurrence of a transition. More specifically, this kind of information flow arises when the occurrence of a transition implies that another transition did not occur or cannot occur in the future. Reveals relation models positive information flow, whereas excludes relation models negative information flow among transitions of a 1-safe Petri net.

The contributions of this paper can be summarised as follows. We introduce the so called maximal-step computation tree which represents behaviour of a 1-safe free-choice net under the maximal-step semantics. We show the correspondence between this tree and the net unfolding. We show that there exists a finite prefix of such tree, named full prefix, on which reveals and excludes relations between the transitions of the corresponding Petri net can be computed efficiently. Thus, the full prefix can be used to determine information flow and to verify noninterference on the basis of reveals and excludes relations as defined in [35], [36]. We prove that the full prefix is sufficient to directly determine reveals relations, and we provide an algorithm to compute excludes relations on it. We consider the notion of footprint, which is the set of transitions observed along a system run. We provide an algorithm to compute footprints of all the maximal runs of a system. The footprints can then be used for further analysis for information flow among sets of transitions. The full prefix contains a summary of the corresponding net's behaviour, and all possible maximal-step computations are deducible from it. Thus, in addition to information-flow analysis, the prefix can be used for the analysis and verification of various behavioural properties of a distributed system modelled by a 1-safe free-choice net.

The paper is organised as follows. Section II provides the necessary background on Petri nets to be reasonably self contained. Section III recalls the definitions of reveals and excludes relations in the context of Petri net information flow. Section IV formalises the maximal-step computation tree and its full prefix. In Section V, the prefix is used for information flow analysis by computing reveals and excludes relations. Section V-C is dedicated to the computation of footprints of all the maximal paths of the maximal-step computation tree from its full prefix. We discuss the use of our notions and methods on some examples in Section VI. Section VII concludes the paper and discusses some possible future work.

II. BACKGROUND IN PETRI NET THEORY

In this section, we give a short introduction to Petri net theory.

A *net* is a triple $N = (P, T, F)$, where P and T are disjoint sets, the elements of P are called *places*, the elements of T are called *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ is called the *flow relation*. The *pre-set* of an element $x \in P \cup T$ is the set ${}^\bullet x = \{y \in P \cup T : (y, x) \in F\}$. The *post-set* of x is the set $x^\bullet = \{y \in P \cup T : (x, y) \in F\}$. Analogously, let $X \subseteq P \cup T$ be a subset of elements. Its pre-set is defined as ${}^\bullet X = \{y \in P \cup T : \exists x \in X : (y, x) \in F\}$, and its post-set is defined as $X^\bullet = \{y \in P \cup T : \exists x \in X : (x, y) \in F\}$.

An (ordinary) Petri net $\Sigma = (P, T, F, m_0)$ is defined by a net (P, T, F), and an initial marking $m_0 : P \to \mathbb{N}$. A net is finite if the sets of places and of transitions are finite.

A *marking* is a map $m : P \to \mathbb{N}$. Markings represent global states of a net. A transition t is *enabled* at a marking m, denoted $m[t\rangle$, if, for each $p \in {}^\bullet t$, $m(p) > 0$. Let t be enabled at m; then, t can *occur* (or *fire*) in m producing the new marking m', denoted $m[t\rangle m'$ and defined as follows:

$$m'(p) = \begin{cases} m(p) - 1 & \text{for all } p \in {}^\bullet t \setminus t^\bullet \\ m(p) + 1 & \text{for all } p \in t^\bullet \setminus {}^\bullet t \\ m(p) & \text{in all other cases.} \end{cases}$$

Example 1: A graphical representation of a Petri net is illustrated in Fig. 1. Conventionally, places are illustrated as circles and transitions are illustrated as squares. The marking is represented through the tokens inside the places, which are illustrated as small black circles. The initial marking m_i of the Petri net in Fig. 1 is $m_i(1) = 1, m_i(2) = 1, m_i(j) = 0$ $\forall j \notin \{1, 2\}$. At the initial marking transitions a, f and c are enabled, i.e., all the preconditions of these transitions are marked. When transition c fires it consumes the token from its precondition, namely place 2, and puts a token to its postcondition, namely place 4. Hence, the marking obtained after the occurrence of c is $m_1(1) = 1, m_1(4) = 1, m_1(j) = 0$, $\forall j \notin \{1, 4\}$.

A marking q is *reachable* from a marking m if there exist transitions $t_1 \dots t_{k+1}$ and intermediate markings $m_1 \dots m_k$

such that: $m[t_1\rangle m_1[t_2\rangle m_2 \ldots m_k[t_{k+1}\rangle q$. The set of markings reachable from m will be denoted by $[m\rangle$. A marking m is safe if $m(p) \in \{0, 1\}$ for all $p \in P$. If all the markings in $[m_0\rangle$ are safe, then $\Sigma = (P, T, F, m_0)$ is said to be *1-safe* (or, shortly, safe). Σ is called *1-live* if for all $t \in T$ there exists $m \in [m_0\rangle$ such that $m[t\rangle$. In the rest of the paper, we will consider systems modelled by 1-safe, 1-live Petri nets such that each transition has a nonempty preset, in which the underlying nets are finite. Since we restrict to 1-safe nets, from now on we will represent markings as subsets of places, where $p \in m$ iff $m(p) = 1$. With this interpretation, a transition t is enabled in a marking m iff ${}^\bullet t \subseteq m$.

Example 2: The net in Fig. 1 is 1-safe, and its initial marking can be equivalently expressed as $\{1, 2\}$. *After the occurrence of transition c, the Petri net reaches the marking* $\{1, 4\}$.

Petri nets allow for representing different kinds of relations between transitions. Given a marking m, two transitions t_1 and t_2 are sequential if the occurrence of t_1 is necessary to enable t_2. In Fig. 1, c and e are sequential, because e is enabled by the occurrence of c. Also, a and b are sequential: even if the occurrence of a may not be sufficient to enable b (since c may fire), the occurrence of a is necessary to enable b. In a marking m, two transitions t_1 and t_2 are in *conflict* if they are both enabled and ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$. In this case, only one of them can fire, because the occurrence of one disables the other. For example, in Fig. 1, a and f are in conflict, because they both need the token from place 1, but only one can consume it. Finally, given a marking m, two transitions t_1 and t_2 are concurrent if they are both enabled in m and they can both fire, no matter in what order. In Fig. 1, c is concurrent with both a and f (although, as we specified before, a and f are not concurrent with each other).

Given a marking m, a *step* is a conflict free subset of transitions that are all enabled at m. A step is *maximal* if it is maximal with respect to set inclusion. In the net in Fig. 1, the maximal steps enabled in the initial markings are $\{a, c\}$ and $\{c, f\}$.

A marking is a *deadlock* if no transition is enabled. Examples of deadlocks are the markings $\{6, 8\}$ or $\{3, 8\}$ in the net of Fig. 1.

Let $N_i = (P_i, T_i, F_i)$ be a net for $i = 1, 2$. A map $\lambda : P_1 \cup T_1 \to P_2 \cup T_2$ is a morphism from N_1 to N_2 if:

1) $\lambda(P_1) \subseteq P_2$; $\lambda(T_1) \subseteq T_2$
2) $\forall t \in T_1$ the restriction of λ to ${}^\bullet t$ is a bijection from ${}^\bullet t$ to ${}^\bullet\lambda(t)$
3) $\forall t \in T_1$ the restriction of λ to $t^\bullet$ is a bijection from $t^\bullet$ to $\lambda(t)^\bullet$

Σ is a *free-choice* net if for all $t_1, t_2 \in T$ such that ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, we have ${}^\bullet t_1 = {}^\bullet t_2$. Historically, nets with this property were called *extended-free-choice*, the term *free-choice* referring to nets where conflicting transitions share exactly one precondition, and have no other preconditions. In *extended-free-choice*, transitions can have an arbitrary number of preconditions, as long as the above property holds. However, it was shown in [8] that these two classes are equivalent, and we will henceforth consider free-choice nets to be those where two transitions sharing at least one precondition are required to share all of their preconditions.

From Section IV on, we will consider free-choice nets as defined above.

We now introduce two technical relation that will be useful to define the idea of *unfolding* of a Petri net. The $\prec$ relation on a net N is the transitive closure of F and $\preceq$ is the reflexive closure of $\prec$. Given two elements $x, y \in P \cup T$, $x\#y$, iff there exist $t_1, t_2 \in T : t_1 \neq t_2, \quad t_1 \preceq x, t_2 \preceq y$ and there exists $p \in {}^\bullet t_1 \cap {}^\bullet t_2$.

A net $N = (B, E, F)$, possibly infinite, is an *occurrence net* if the following restrictions hold:

1) $\forall x \in B \cup E : \neg(x \prec x)$
2) $\forall x \in B \cup E : \neg(x\#x)$
3) $\forall e \in E : \{x \in B \cup E \mid x \preceq e\}$ is finite
4) $\forall b \in B : |{}^\bullet b| \leq 1$

In an occurrence net, the elements of B are called *conditions* and the elements of E are called *events*; the transitive and reflexive closure of F forms a *partial order*. The set of minimal elements of an occurrence net N with respect to $\preceq$ will be denoted by $\min(N)$. Since we only consider the nets in which every transition has a nonempty preset, the elements of $\min(N)$ are conditions.

An occurrence net represents the alternative histories of a system; therefore its underlying graph is acyclic (cond. 1), and paths branching from a condition, corresponding to a choice between alternative behaviours, never converge (cond. 2). A *configuration* of an occurrence net $N = (B, E, F)$ is a set C of events which is causally closed and free of conflicts. C is *maximal* if it is maximal with respect to set inclusion.

A *B-cut* of N is a maximal set of pairwise concurrent elements of B, and can be intuitively seen as a global state of the net in a certain moment. An *E-cut* of N is a maximal set of pairwise concurrent elements of E, that corresponds with a maximal step on N. By analogy with net systems, we will sometimes say that an event e of an occurrence net is enabled at a B-cut β, denoted $\beta[e\rangle$, if ${}^\bullet e \subseteq \beta$. A B-cut is a deadlock if no event is enabled at it. We will denote by $C(\gamma)$ the set of all events in the causal closure of a B-cut γ, i.e., $C(\gamma) = \{e \in E \mid e \prec b,\ b \in \gamma\}$

Example 3: In Fig. 1, $\mathrm{unf}(\Sigma)$ *is an occurrence net. The set* $C_1 = \{f_1, c_1, e_1, c_2, h_2\}$ *is a maximal configuration of* $\mathrm{unf}(\Sigma)$. *The sets* $\{1_1, 2_1\}$ *and* $\{3_1, 2_2\}$ *are examples of B-cuts, whereas* $\{a_1, c_1\}$ *and* $\{b_2\}$ *are examples of E-cuts. In particular,* $\{a_1, c_1\}$ *is a maximal step enabled in* $\{1_1, 2_1\}$, *whereas* $\{b_2\}$ *is a maximal step enabled in* $\{3_1, 2_2\}$. *The B-cut* $\gamma = \{6_1, 8_2\}$ *is a deadlock, and* $C(\gamma) = C_1$.

A *branching process* of a Petri net $\Sigma = (P, T, F, m_0)$ is a pair (O, λ), where $O = (B, E, F)$ is an occurrence net, and λ is a morphism from O to Σ such that:

1) $\forall p \in P\ m_0(p) = |\lambda^{-1}(p) \cap \min(O)|$
2) $\forall x, y \in E$, if ${}^\bullet x = {}^\bullet y$ and $\lambda(x) = \lambda(y)$, then $x = y$

We extend the definition of λ to the set of configurations on the branching process: for each configuration C, for each $e_i \in C$, $\lambda(C) = \bigcup_i \lambda(e_i)$. The set $\lambda(C)$ is the *footprint* of C.

A branching process $\Pi_1 = (O_1, \lambda_1)$ is a *prefix* of $\Pi_2 = (O_2, \lambda_2)$ if there is an injective morphism f from O_1 to O_2 which is a bijection when restricted to $\min(O_1)$, and such that $\lambda_1 = \lambda_2 f$.

Any finite Petri net $\Sigma = (P, T, F, m_0)$ has a unique branching process which is maximal with respect to the prefix relation. This maximal process, called the *unfolding* of Σ, will be denoted by $\text{unf}(\Sigma) = ((B, E, F), \lambda)$, where λ is the morphism from (B, E, F) to (P, T, F) [15].

A *run* is the subnet of an unfolding that is induced by a configuration. A run is *maximal* if the corresponding configuration is maximal.

Example 4: In Fig. 1, $\text{unf}(\Sigma)$ *is the unfolding of* Σ. *A maximal run of* $\text{unf}(\Sigma)$ *is the subnet such that: the set of events is the configuration* $C_1 = \{f_1, c_1, e_1, c_2, h_2\}$, *the set of corresponding conditions is* $B_1 = \{1_1, 6_1, 2_1, 4_1, 2_2, 4_2, 8_2\}$ *and F is the flow relation on* $\text{unf}(\Sigma)$ *restricted to the elements in* $B_1 \cup C_1$. *The footprint of* C_1 *is* $\lambda(C_1) = \{f, c, e, h\}$.

III. INFORMATION FLOW BETWEEN TRANSITIONS OF A PETRI NET

Consider a concurrent system which has two distinct kinds of actions: *high* (hidden) and *low* (observable). The high actions are confidential, whereas the low ones are not. It is important that an attacker or an observer with no authorisation cannot deduce information about the high actions. Noninterference formally guarantees that this kind of information flow does not happen, so that the system is secure.

Petri nets can model concurrency, conflict and causal dependency. However, these are not the only meaningful logical relations between transitions of a Petri net when information-flow security is considered. Two transitions which are not related to each other in terms of above mentioned relations can still be related in the sense that the occurrence of one gives information about the occurrence of the other. For example, the occurrence of an observable transition might imply the occurrence or nonoccurrence of a hidden transition, endangering information-flow security.

In [36] reveals, extended-reveals and excludes relations were introduced for the transitions of a Petri net with the aim of modelling information flow in concurrent systems. A number of noninterference notions were defined by means of reveals and excludes relations providing different levels of security guarantees. Below we recall these relations. For all, we assume *progress* of the system, i.e., an enabled transition either fires or gets disabled by another transition that is in conflict.

A. REVEALS RELATION

The reveals relation was originally introduced for events of an occurrence net in [37]. In [38] the authors applied it in the field of fault diagnosis. In [36] reveals relation is redefined for the transitions of a 1-live Petri net in order to express positive information flow. In this paper we only consider 1-safe nets but reveals relation between transitions is actually defined without any restriction on safeness.

We say transition t_1 reveals transition t_2 if each maximal configuration which contains an occurrence of t_1 also contains at least one occurrence of t_2. This means that, from the occurrence of t_1, we can deduce that t_2 has already occurred or will occur inevitably. In other words, the occurrence of t_1 implies the occurrence of t_2 either in the past or in the future. It can violate security if a low transition reveals a high transition.

Definition 1: Let $\Sigma = (P, T, F, m_0)$ *be a 1-safe Petri net,* C_{max} *the set of all its maximal configurations and* $t_1, t_2 \in T$ *be two transitions. Then* t_1 reveals t_2, *denoted* $t_1 \rhd t_2$, *iff* $\forall\, C \in C_{max}\ \ t_1 \in \lambda(C) \implies t_2 \in \lambda(C)$.

Reveals is a reflexive and transitive relation. However, it is neither symmetric nor anti-symmetric.

Example 5: In the net in Fig. 1, $f \rhd c$, *whereas* $c \not\rhd f$, *since there is a maximal configuration of the unfolding with footprint* $\{a, c, h\}$. *Some other reveals relations are* $b \rhd a$, $b \rhd d$, $h \rhd c$

B. EXTENDED-REVEALS RELATION

In some cases, one transition alone does not give much information about the occurrence of another transition, however, the occurrence of a set of transitions might give information about the occurrence of others. For example, the occurrence of all the transitions in set W can imply the occurrence of some transitions in set Z. This relation was originally defined in [39] for the events of an occurrence net and used in the field of fault diagnosis. Later the relation was expressed in terms of the transitions of a Petri net [36], and used to analyse noninterference for information-flow security. Depending on the chosen noninterference notion, it can violate security if a group of low transitions extended-reveals a group of high transitions.

Definition 2: Let $W, Z \subseteq T$ *and* C_{max} *be the set of all maximal configurations. Then* W extended-reveals Z, *denoted* $W \twoheadrightarrow Z$ *iff* $\forall\, C \in C_{max}$

$$\bigwedge_{t_1 \in W} t_1 \in \lambda(C) \implies \bigvee_{t \in Z} t_2 \in \lambda(C)$$

Extended-reveals relation is reflexive and nontransitive. It is neither symmetric nor anti-symmetric. The reveals relation coincides with the extended-reveals relation between singletons, i.e., $t_1 \rhd t_2$ can be written as $\{t_1\} \twoheadrightarrow \{t_2\}$.

Example 6: In the net in Fig. 2, although e reveals both b and c, neither b nor c reveals e alone. However, if both b and c occur, then e will occur inevitably because of the progress assumption. So we can write $\{b, c\} \twoheadrightarrow \{e\}$. *Similarly, both f and g reveal d but d reveals neither f nor g. However, the occurrence of d implies that either f or g will occur inevitably. This can be expressed by the extended-reveals relation, i.e.,* $\{d\} \twoheadrightarrow \{f, g\}$.

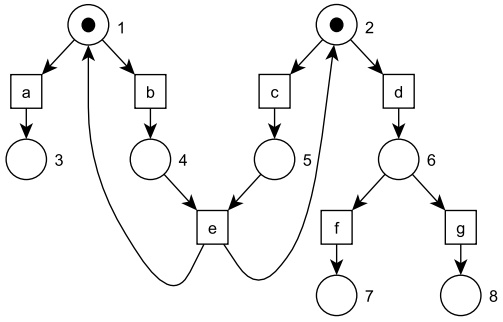


FIGURE 2. A 1-safe free-choice net.

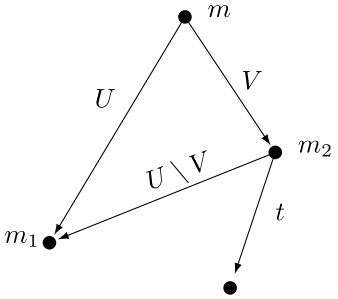


FIGURE 3. Proof of Lemma 1.

C. EXCLUDES RELATION

Excludes relation was also introduced in [36] for transitions of a 1-live Petri net in order to express negative information flow. In this paper, we focus on the case of 1-safe Petri nets. Two transitions exclude each other if they never occur in the same run. This means that the occurrence of one implies nonoccurrence of the other. Depending on the chosen noninterference notion, it can violate security if a low transition excludes a high transition.

Definition 3: Let $\Sigma = (P, T, F, m_0)$ *be a 1-safe Petri net,* C_{max} *the set of all its maximal configurations and let* $t_1, t_2 \in T$. t_1 excludes t_2, *denoted* t_1 ***ex*** t_2, *iff* $\forall\, C \in C_{max}\ \ t_1 \in \lambda(C) \implies t_2 \notin \lambda(C)$.

By definition, excludes is a symmetric relation. It is neither transitive nor reflexive. Moreover, it does not coincide with the conflict relation. Transitions which are in conflict at a reachable marking can still appear in the same maximal run, so they may not exclude each other.

Example 7: In the net in Fig. 1, b is in conflict with c, however, the two transitions are not in exclude relation, since there is a maximal configuration with footprint $\{a, b, c, e, d, g\}$. *In the same net,* f ***ex*** b, *since a maximal configuration displaying* f *has either* $\{f, c, e\}$ *or* $\{f, c, e, h\}$ *as footprint.*

IV. MAXIMAL-STEP SEMANTICS ON PETRI NETS

In Section II we have given the definition of maximal step: a set of conflict-free transitions, all enabled at a marking m. Transitions belonging to a step can fire concurrently without interfering with one another.

The behaviour of a Petri net is commonly expressed by means of its marking graph, which displays the firings of single transitions. In some applications, however, the so-called *maximal-step semantics* is used. Here, the set of reachable markings is computed by firing only maximal steps.

For general 1-safe Petri nets, the two semantics produce different sets of reachable markings; more precisely, any reachable marking under the maximal-step semantics is also a reachable marking for the ordinary semantics, but not vice versa. For example, in Fig. 1 the marking $\{2, 3\}$ is not reachable for maximal-step semantics, but it is reachable under the ordinary semantics. This difference has a counterpart on the set of runs of the unfolding of a net: some maximal runs of the unfolding of a net system are not maximal when considering maximal-step semantics.

For free-choice nets, it is still true that the sets of reachable markings differ, but the sets of maximal runs in the unfolding coincide, as shown below.

A. THE TREE OF MAXIMAL-STEP COMPUTATIONS

In this section we define a tree structure that represents all computations of a free-choice net Σ according to maximal-step semantics. We show that this model is semantically equivalent to the unfolding for free-choice nets (this does not hold for more general classes of Petri nets).

The equivalence of semantics is a corollary of the following lemma.

Lemma 1: Let $\Sigma = (P, T, F, m_0)$ *be a 1-safe, free-choice Petri net. Let* $m \in [m_0\rangle$ *be a reachable marking of* Σ, $U \subseteq T$ *be a maximal step enabled at m, with* $m[U\rangle m_1$, *and* $V \subseteq U$. *Then V is enabled at m. Suppose* $m[V\rangle m_2$. *If* $m_2[t\rangle$, *then either* $m[t\rangle$ *or* $m_1[t\rangle$.

Proof: Suppose that t is not enabled at m. Then the pre-set of t can be split in two disjoint subsets, A and B, with $A \subseteq m$ and $B \subseteq V^\bullet$. In words, an input place of t either belongs to m or is marked by a transition in V. All places in B belong to m_1. Suppose that there is a place p in A which is not in m_1. Then, there must be a transition t_2 in $U \setminus V$ which takes the token from p. Then t and t_2 have a common input place; since the net is free-choice, this implies that t is enabled at m, contradicting the hypothesis.

The proof is illustrated in Fig. 3 □

The tree of maximal-step computations of Σ (msct(Σ)) is a labelled tree, defined inductively. Each node is associated to a B-cut of unf(Σ) and each arc is labelled by an E-cut of unf(Σ) which corresponds to a maximal step of Σ.

Definition 4: The tree of maximal-step computations *of* Σ, *denoted by* msct(Σ), *is a labelled rooted tree defined by the following clauses.*

- *The root of* msct(Σ) *is the initial cut of* unf(Σ).
- *If* γ *is a node of the tree, and* V *is an E-cut such that* $^\bullet V \subseteq \gamma$, *then the cut* γ' *that is reached from* γ *when* V *occurs is a node of the tree, and a child of*

γ; $\gamma' = (\gamma \setminus {}^\bullet V) \cup V^\bullet$. *The edge between* γ *and* γ' *is labelled with* V.
- *Nothing else is a node of the tree.*

A *path* in the msc-tree from a node γ_0 is a sequence of nodes, $\gamma_0\gamma_1\cdots$ such that, for each $i > 0$, γ_i is a child of γ_{i-1}. A path is maximal in the msc-tree if it starts from the root and cannot be extended. Given two finite paths $\pi_1 = \gamma_0\gamma_1\ldots\gamma_n$ and $\pi_2 = \gamma_n\gamma_{n+1}\ldots\gamma_m$ such that the last node of π_1 is equal to the first node of π_2, we define their concatenation as the path $\pi_1 \cdot \pi_2 = \gamma_0\gamma_1\ldots\gamma_m$. The tree induces a partial order between its nodes and arcs. Let γ_1 and γ_2 be two nodes of the msc-tree. $\gamma_1 < \gamma_2$ iff there is a path $\pi = \gamma_1\ldots\gamma_2$ starting in γ_1 and ending in γ_2.

In msct(Σ) each path π starting from the root is associated to a configuration of unf(Σ); the configuration is obtained by taking the union of the labels of the arcs along the path. Clearly this is a configuration as defined in Section II, and will be denoted by conf(π). λ(conf(π)) denotes the footprint of the path π.

The correspondence between paths and configurations is not bijective, since a configuration can include only some of the events forming a maximal step at a given B-cut. However, the correspondence is bijective for maximal paths and maximal configurations, as stated in the following.

Lemma 2: Let C *be a maximal configuration of* unf(Σ). *Then there is a maximal path* π *of* msct(Σ), *such that* $C =$ conf(π).

Let π *be a maximal path in* msct(Σ); *then* conf(π) *is a maximal configuration of* unf(Σ).

Proof: Let C be a maximal configuration of unf(Σ). The set of events in C inherits from unf(Σ) a partial order. Let C_1 be the set of minimal elements of this partial order; then C_1 is an E-cut, a maximal step enabled at the initial B-cut of unf(Σ), corresponding to a maximal step enabled at the initial marking of Σ; let γ_1 be the B-cut reached by firing the events in C_1. In msct(Σ) there is an arc, labelled by C_1, from the root to the node corresponding to γ_1. Now take the minimal elements in $C \setminus C_1$; these events form a maximal step enabled at γ_1, leading to a B-cut γ_2, and there is a corresponding arc in msct(Σ), from γ_1 to γ_2. By iterating this procedure, we construct a path of msct(Σ). The maximal configuration C can be decomposed into a (possibly infinite) sequence of steps $C_1C_2\cdots$, giving a maximal, corresponding path in msct(Σ).

Let now π be a maximal path in msct(Σ). By construction of π, the union of the arc labels along π is a configuration of unf(Σ). Let conf(π) be this configuration, and suppose conf(π) is not maximal; then, there is an event e which is not in conf(π), and such that conf(π) $\cup \{e\}$ is a configuration. Then e cannot be in conflict with any event in conf(π). There must be a cut γ, associated to a node of msct(Σ), such that e is enabled by γ, and γ is the first cut enabling e along π. Let H be the step labelling the arc going out of γ along π. Since no event in H is in conflict with e, $H \cup \{e\}$ is a step at γ, contradicting maximality of H. ■

Example 8: In Fig. 4, Σ_1 *is a 1-safe free-choice net. Both the unfolding,* unf(Σ_1), *and the maximal-step computation tree,* msct(Σ_1), *are illustrated in the figure.*

The initial marking of the net is {1}. *In the unfolding, it corresponds to* min(unf(Σ_1)), *which is the B-cut consisting of a condition labelled by* 1_1. *In the tree, it corresponds to the node labelled by* 1_1 *which is also the root of the tree. At the initial marking, transitions a and b are enabled. Firing of b leads the system to a deadlock. We can see a maximal configuration in the unfolding and a maximal path in the tree representing the occurrence of b. The firing of a from the initial marking leads the system to marking* $\{3, 4\}$. *This marking is represented in the unfolding as a B-cut and it is represented in the tree as a node labelled by* $3_1, 4_1$. *In the unfolding,* c_1 *and* d_1 *are two concurrent events which can occur in any order. In case* c_1 *occurs first,* $\{5_1, 4_1\}$ *is reached. In case* d_1 *occurs first,* $\{3_1, 6_1\}$ *is reached. Occurrence of both of these events leads to* $\{5_1, 6_1\}$. *In the tree, only the maximal steps are considered, so* c_1, d_1 *occur together and the node* $\{5_1, 6_1\}$ *is reached. There are no correspondences of the B-cuts* $\{5_1, 4_1\}$ *and* $\{3_1, 6_1\}$ *in the tree.*

Although some reachable markings are not represented in the tree, each maximal configuration of the unfolding correspond to a maximal path in the tree and vice versa.

Lemma 2 does not hold in general for the nets that are not free-choice.

Example 9: The net in Fig. 1 is not free choice, because ${}^\bullet b \cap {}^\bullet c \neq \emptyset$, *and* $3 \in {}^\bullet b \wedge 3 \notin {}^\bullet c$. *In the initial marking, the maximal steps are* $\{a, c\}$ *and* $\{f, c\}$, *therefore, by considering only maximal steps we would assume that c must belong to all the maximal configurations of the unfolding. This is not true, since there is a maximal configuration with footprint* $\{a, b, d, g\}$, *in other words, there exist a maximal configuration in which c is not in the set of labels.*

Lemma 3: Let γ_1 *and* γ_2 *be two nodes in* msct(Σ), *such that* $\lambda(\gamma_1) = \lambda(\gamma_2)$. *Then the two sub-trees of* msct(Σ) *having* γ_1 *and* γ_2 *as roots (and as nodes all their descendants in* msct(Σ)*) are isomorphic.*

Proof: The proof is directly derived from the definition of msct(Σ) and properties of unfoldings (see [15], [16]). ■

B. SOME NOTATION FOR MAXIMAL-STEP COMPUTATION TREE

In the following we will introduce some msc-tree related notations which then will be used throughout the rest of the paper. In all the following items, A will denote a msc-tree, or any sub-tree of it.

- Π_A is the set of all the maximal paths of A.
- Given a node γ in a tree A, let A_γ denote the largest sub-tree of A whose root is γ.
- Given a transition a, we write

$$\Pi_A^a := \bigcup_{\lambda(e)=a} \{\pi \in \Pi_A \mid e \in \text{conf}(\pi)\}$$

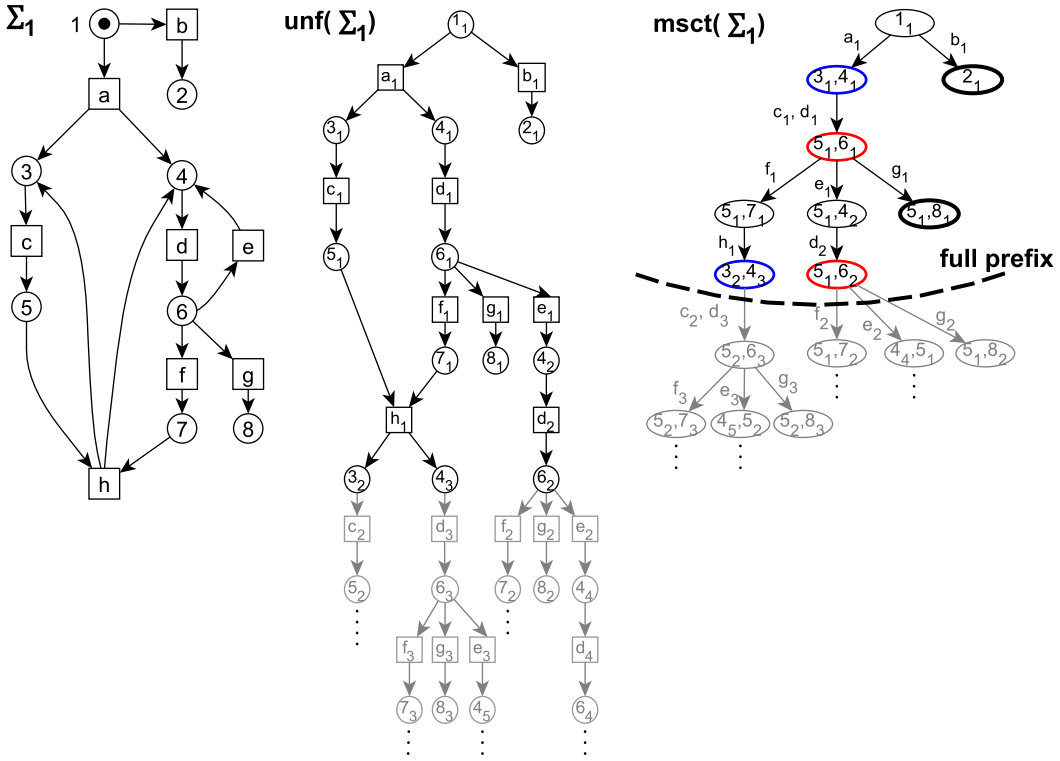


FIGURE 4. A 1-safe free-choice net, its unfolding and its maximal-step computation tree.

- Given a path π of A, and a node $\gamma \in \pi$, let $\pi \downarrow \gamma$ be the initial segment of π whose last node is γ, and $\gamma \uparrow \pi := \pi \cap A_\gamma$
- Let π and π' be two paths, let γ_1, γ_2 and γ_1', γ_2', be their respective two first nodes. We say $\pi \simeq \pi'$ iff $\lambda(\gamma_1) = \lambda(\gamma_1')$, the edges (γ_1, γ_2) and (γ_1', γ_2') are respectively labelled with steps V_1 and V_2 such that $\lambda(V_1) = \lambda(V_2)$, and $\gamma_2 \uparrow \pi \simeq \gamma_2' \uparrow \pi'$.
- Given a finite path π of A, let leaf(π) denote the single leaf in this path.

C. A FINITE PREFIX OF MAXIMAL-STEP COMPUTATION TREE

We now define a finite prefix of msct(Σ), which contains the information needed to determine the whole *reveals* relation in Σ. The prefix is built starting from the root, and adding nodes along a path until we meet a B-cut γ such that $\lambda(\gamma) = \lambda(\gamma')$, for some γ' on the path between the root and γ.

Definition 5: Let Σ be a 1-safe free-choice net. The full prefix *of* msct(Σ)*, denoted* fp(Σ)*, is a labelled rooted tree defined by the following clauses.*

1) *The root of* fp(Σ) *is the root of* msct(Σ).
2) *Let γ be a node of* fp(Σ)*. If there is no γ' such that $\gamma' < \gamma$ and $\lambda(\gamma') = \lambda(\gamma)$, then all the children of γ in the* msct(Σ) *are nodes in* fp(Σ)*; otherwise γ is a leaf in* fp(Σ).

Given a leaf l of the full prefix, let rep(l) denote the ancestor of l corresponding to the same marking. If l corresponds to a deadlock, then rep(l) is undefined.

Example 10: In Fig. 4, the border of the full prefix is shown via a dashed line. The corresponding part of the unfolding is drawn in black whereas the rest is drawn in pale grey. The nodes of the tree which correspond to deadlocks are in bold black. If the leaf does not correspond to a deadlock, then it corresponds to a marking which is already reached in the same path before. The two corresponding nodes are drawn in the same colour. For example, the path in which a_1, c_1, d_1, f_1, h_1 occur reaches the marking $\{3, 4\}$ twice. This path is maximal in the full prefix. In total, there are four maximal paths in the full prefix of msct(Σ_1)*. In the following sections we will show that all the reveals and excludes relations of the net Σ_1 can be computed on the full prefix.*

The full prefix of a msc-tree of a Petri net can be constructed on the basis of Definition 4 and Definition 5. Note that this construction can be done directly from the Petri net without resorting to the unfolding.

Lemma 4: The full prefix of a msc*-tree is finite.*

Proof: Since the number of transitions is finite, for each node the number of children is finite. Let n be the total number of reachable markings in the net, no path of the tree can have more than $n + 1$ nodes without reaching a marking that was already visited. ■

Lemma 5: For each maximal path π of a msc*-tree, there is a maximal path π' of its full prefix, such that π' is a prefix of π.*

Proof: Let π be a maximal path of an msc-tree and γ be the last node of the intersection between π and the full prefix. γ cannot have any child in the prefix, otherwise one of them should also be in the intersection, since by definition, if γ is not a leaf, all the children in the msc-tree are also in its full prefix. Hence γ must be a leaf, and the path from the root to γ is maximal in the prefix. ∎

Next, we describe an operation involving both msc-tree and full prefix, that is at the core of the main results of this paper. In particular, it will allow us to show that the full prefix gathers enough information to directly determine some of the information flow in the system. It is fundamentally adapted from [38] to the structures here at stake. Informally, the idea is that whenever a path contains two nodes representing the same marking, then we may obtain a new path by collapsing the whole segment between these two nodes to a single node representing that marking.

Definition 6 (Peeling): Let A be a msc*-tree. We define a function* peel $: \Pi_A \to \Pi_A$. *Given a maximal path π of A,* peel(π) *is called the* peeling *of π.*

Consider $\pi' = \pi \cap L$. From Lemma 5, $\pi' \in \Pi_L$. If leaf(π') *is a deadlock, then* peel$(\pi) := \pi$. *Otherwise,* rep(leaf(π')) *is well-defined, and there is a single path* peel(π) *satisfying that* $\pi \downarrow \text{rep}(\text{leaf}(\pi')) = \text{peel}(\pi) \downarrow \text{rep}(\text{leaf}(\pi'))$ *and* $\text{leaf}(\pi') \uparrow \pi \simeq \text{rep}(\text{leaf}(\pi')) \uparrow \text{peel}(\pi)$.

In terms of Petri nets, such a path π is nothing but a run which may cycle back to some previously visited marking. Then peel(π) and π corresponds to the same run but for skipping the cycle altogether.

Lemma 6: Let Σ be a free-choice Petri net and fp(Σ) *the full prefix of* msct(Σ). *For each node γ in* msct(Σ), *there is at least a node γ' in* fp(Σ) *such that $\lambda(\gamma) = \lambda(\gamma')$.*

Proof: Let γ be a node of msct(Σ). If γ is also in fp(Σ), the proof is trivial. Otherwise, γ must follow a leaf in fp(Σ), since by definition, for each γ_i internal node on fp(Σ), for each child γ_{i+1} of γ_i in msct(Σ), γ_{i+1} is also in fp(Σ). Let γ_j be the leaf preceding γ and π a maximal path on msct(Σ) including γ. By construction, there must be a node $\gamma'_j < \gamma_j$ such that $\lambda(\gamma_j) = \lambda(\gamma'_j)$. We can peel π by identifying γ'_j with γ_j. In the peeled path π', let γ' be the cut associated to γ by the peeling operation. There are two cases: (1) there is no pair of markings γ_1, γ'_1 such that $\lambda(\gamma_1) = \lambda(\gamma'_1)$ and $\gamma_1 < \gamma'_1 < \gamma'$. Then, by construction γ' is in fp(Σ). (2) γ' follows two cuts γ_1 and γ'_1 with $\lambda(\gamma_1) = \lambda(\gamma'_1)$. In this case, we can repeat the peeling procedure until a node associated with γ is in fp(Σ). This will happen in a finite number of steps, since each time the number of nodes between the node associated with γ and the root decreases. ∎

Remark 1: If γ is not a deadlock, at least an occurrence of γ' as described in Lemma 6 is an internal node. In order to see it, we can observe that if γ' is a leaf and it is not a deadlock, then there is another node in fp(Σ) *preceding γ' and associated to the same marking by construction.*

A consequence of Lemma 6 is that at least an occurrence of all the transitions in Σ appears in fp(Σ): let t be a transition in Σ, an occurrence of t must appear also in msct(Σ). Let γ be a node in msct(Σ) and V the label of an outgoing arc from γ such that $t \in \lambda(V)$; by Lemma 6, there is a node γ' in fp(Σ) corresponding to the same marking, and an outgoing arc from γ labelled with V', such that $\lambda(V) = \lambda(V')$.

We now define an extension operation on finite paths.

Definition 7: Let $\pi = \gamma_0\gamma_1 \dots \gamma_n$ be a finite path in the msc*-tree A, and L be the full prefix of A. For each internal node $\gamma \in L$ such that $\lambda(\gamma) = \lambda(\gamma_n)$ and for each $\pi' \in \Pi_{L_\gamma}$, we define the* extension *of π through π' as the function* $\text{ext}(\pi, \pi') = \pi \cdot \pi_{ex}$, *where π_{ex} is the path with first node γ_n and such that $\pi_{ex} \simeq \pi'$.*

The existence of π_{ex} in Definition 7 is a direct consequence of Lemma 3. The extension operation allows us to state another property of the full prefix, that establish a relation between its paths and the paths of the msc-tree. This is expressed by the following lemma.

Lemma 7: For each π finite path in the msc*-tree, if π is not contained in* fp(Σ), *then we can find a path including it, by starting from a maximal path of the full prefix and applying recursively the extension operation.*

Proof: Consequence of Lemma 3, Lemma 6 and Remark 1. ∎

The result of the previous Lemma applies also to infinite paths when considering the limit of applying recursively the extension operation. This means that we can extract all possible system behaviour under maximal-step semantics from the full prefix.

V. INFORMATION-FLOW ANALYSIS ON THE FULL PREFIX

In this section we study the relations that are recalled in Section III on the full prefix. We show that the full prefix is an adequate basis for computing these relations and so it can be used to verify information-flow security with the noninterference notions introduced in [36].

The algorithms provided in this section require a previous computation of the full prefix, and their running time depends on the number of distinct transitions of the system, as well as on the size of the full prefix. It is therefore noteworthy that the length of the full prefix is at most the number of markings of the system that are reached under maximal-step semantics, plus one repeated marking. In case of paths composed of many cycles, the path will be as large as the least common multiple of the length of these cycles, since after this number of maximal steps, each cycle will be back at its initial conditions. In each node, the number of children depends on the number of transitions concurrent and in conflict in the corresponding marking. In particular, if the marking m enables n groups of concurrent transitions, each of them with k_i, $i \in \{1, \dots, n\}$, transitions in conflict, then the node associated with the marking m will have $\prod_{i=1}^{n} k_i$ children. Once the full prefix is computed, it can serve all the subsequent information flow analysis, as presented next.

A. COMPUTING REVEALS ON THE FULL PREFIX

Let a and b be two transitions of Σ, we say that a reveals b if and only if each maximal configuration which includes an occurrence of a also includes an occurrence of b (see Definition 1). This is directly translated to the following due to the equivalence of maximal configurations of an unfolding and maximal paths of the corresponding msc-tree. a reveals b if and only if each maximal path of msct(Σ) which includes an occurrence of a also includes an occurrence of b. In other words, let A be msct(Σ), $a \rhd b$ iff $\Pi_A^a \subseteq \Pi_A^b$.

In the following example, we study the reveals relation among transitions of a Petri net on the full prefix of its msc-tree.

Example 11: In Fig. 4, in the net Σ_1, a reveals both c and d. Similarly, h reveals a, c, d and f. In fact, each maximal path of the full prefix of msct(Σ_1) *that contains a also contains c and d. And each maximal path of the full prefix which contains h also contains a, c, d and f. Looking at the other direction, we see that each maximal path of the full prefix which contains e also contains d and in fact e reveals d in the net system. However, the converse is not true. There is a maximal path of the full prefix which contains d but does not contain e (namely the maximal path which ends with node $\{3_2, 4_3\}$) and in fact d does not reveal e in the net system.*

Next, we prove that, for 1-safe free-choice nets, the full prefix is enough to compute the reveals relation. In other words, if $a \rhd b$ in Σ, then each maximal path of fp(Σ) which has an occurrence of a also has an occurrence of b and vice versa.

Theorem 1: Let $\Sigma = (P, T, F, m_0)$ be a 1-safe free-choice Petri net, $a, b \in T$, A be the msct(Σ)*, and L be its full prefix,* fp(Σ). $\Pi_A^a \subseteq \Pi_A^b \Leftrightarrow \Pi_L^a \subseteq \Pi_L^b$.

Proof: We use proof by contraposition, so we first show that if $\Pi_L^a \not\subseteq \Pi_L^b$ then $\Pi_A^a \not\subseteq \Pi_A^b$. Let ρ be a maximal path in L such that $\rho \in \Pi_L^a$ and $\rho \notin \Pi_L^b$. If $\lambda(\text{leaf}(\rho))$ is a deadlock the conclusion is trivial because ρ is a maximal path in A. Otherwise, rep(leaf(ρ)) exists in ρ, and we can extend infinitely many times ρ with an isomorphic copy of the segment between rep(leaf(ρ)) and leaf(ρ). This procedure produces a maximal path in A with the same labels of ρ, therefore without occurrences of b.

We now suppose that $\Pi_A^a \not\subseteq \Pi_A^b$ and we show that $\Pi_L^a \not\subseteq \Pi_L^b$. The proof proceeds with an iterative procedure meant to find a maximal path of the prefix with an occurrence of a and no occurrence of b, thus showing that a does not reveal b in L. Informally, this procedure constructs, at each step, a maximal path of A with an occurrence of a and no occurrence of b, in such a way that the number of elements in the past of its first occurrence of a is reduced at each step. Since there can only be finitely many such elements, the procedure will reach a maximal path of L containing an occurrence of a but not of b, in a finite number of steps.

By hypothesis, there is a maximal path $\rho' \in \Pi_A : \rho' \in \Pi_A^a \wedge \rho' \notin \Pi_A^b$. By Lemma 5, there is $\rho \in \Pi_L$ that is a prefix of ρ'. Note that ρ has no occurrence of b —otherwise, so would ρ'—, and so if it has an occurrence of a, then $\rho \in \Pi_L^a$ and $\rho \notin \Pi_L^b$. Now, suppose that a does not occur in ρ, and $\gamma_f = \text{leaf}(\rho)$. If γ_f were a deadlock, we would necessarily have that $\rho = \rho'$, but this is a contradiction, since a occurs in ρ' and not in ρ. Then rep(γ_f) is defined, there is only a finite number of elements in ρ' which are in the past of a, and at least one of these must be found between rep(γ_f) and γ_f, not to contradict maximal-step semantics. The path $\rho'' = \text{peel}(\rho')$ is maximal in A, and by Lemma 5, there is a maximal path $\rho^1 \in \Pi_L$ which is a prefix of it. Clearly, b does not occur in ρ'' or ρ^1, and so if a occurs in ρ^1 we reach the desired conclusion. Otherwise, consider leaf(ρ^1). Clearly, it is not a deadlock, and by construction, rep(leaf(ρ^1)) is defined. Since the number of elements between the root and the first occurrence of a in ρ'' is strictly less than that between the root and the first occurrence of a in ρ', we may iterate the procedure by finding prefixes $\rho^2, \ldots, \rho^n$ with no occurrences of b, until eventually ρ^n will have an occurrence of a, for a finite n. ■

With the above proof, we conclude that all the reveals relations among the transitions of Σ can be found by computing the footprints of the maximal paths of fp(Σ) and checking for the corresponding labels among them. Note that maximal paths of fp(Σ) are finite, whereas maximal configurations of the unfolding are in general infinite.

B. COMPUTING EXCLUDES ON THE FULL PREFIX

Let a and b be two transitions of Σ, we say that a excludes b (or b excludes a by symmetry of the relation) if, and only if, no maximal configuration of unf(Σ) has both an occurrence of a and of b (see Definition 3). Due to the equivalence of maximal configurations of an unfolding and maximal paths of the corresponding msc-tree, this is directly translated to the following: a excludes b if, and only if, no maximal path of msct(Σ) displays both a and b. In other words, let A be msct(Σ), a ***ex*** b iff $\Pi_A^a \cap \Pi_A^b = \emptyset$.

In the following example, we will study the excludes relation among transitions of a Petri net on the full prefix of its msc-tree.

Example 12: In the net in Fig. 4, b excludes all the other transitions of the net. By symmetry of the excludes relation, each transition (except b itself) excludes b. These are the only excludes relations in the net.

Now let us examine the full prefix, fp(Σ_1)*, and consider the transitions a and b. The initial node corresponds to the marking* (1)*. From the initial node we can either continue with an occurrence of a or an occurrence of b (and these are the only occurrences of a and b in* fp(Σ_1)*). None of the maximal paths of* fp(Σ_1) *has occurrences of a and b together. However, we cannot conclude a excludes b by looking at the maximal paths of the full prefix. We need to make sure the maximal paths of the* msct(Σ_1) *don't have the two transitions together. To do this, we do not need to construct the whole tree. We can use the full prefix as a basis to decide excludes in a finite number of steps. This procedure is formally defined later in this section.*

The maximal path of $\mathrm{fp}(\Sigma_1)$ *with the occurrence of b ends in a deadlock so it is maximal in the* $\mathrm{msct}(\Sigma_1)$ *as well. It is clear that after the occurrence of b, a cannot occur. Now we need to check if b can occur after a does. We can extend the paths by gluing the nodes corresponding to the same marking. In this way, we can build maximal paths of* $\mathrm{msct}\,\Sigma_1$. *For example, in the maximal path in which* $a, \{c, d\}, f$, *and h occur (let's call it* π_1*), the blue nodes correspond to a repeated marking, namely* $\{3, 4\}$. *So,* π_1 *can be extended in many different ways by gluing the blue nodes and repeating the gluing operation on different repeated marking pairs. It can be extended to a maximal path in which only* a, c, d, f, h, *occur. It can be extended so as to also have occurrences of either e or g or both. But none of the continuations can have an occurrence of b because the marking which enables b can never be reached during this operation. So, we can conclude a excludes b.*

Now let's consider the transitions f and h. We can easily see that there is a maximal path of $\mathrm{fp}(\Sigma_1)$ *in which both occur, namely* π_1. *So we can decide f does not exclude h on the full prefix, without any further computation.*

If we look at the transitions e and g, we can see that the maximal path of $\mathrm{fp}(\Sigma_1)$*, in which e occurs, ends in a node corresponding to the marking* $\{5, 6\}$*, which is repeated in the path. So the path can be extended so that after e occurs, g does as well. This path witnesses that e does not exclude g.*

In the following we present a finite algorithmic procedure to check whether a excludes b on the full prefix of the msc-tree; the pseudocode of this procedure is presented in Algorithm 1. Theorem 2 then proves the correctness of the algorithm.

Algorithm 1 takes the full prefix L and two transitions $a, b \in T$ as input. During its execution, it checks whether there can be an occurrence of b following or being in the step label as an occurrence of a, in any path of the msc-tree. If this is the case, the algorithm returns false, and the procedure terminates. Otherwise, it returns true. In this second case, in order to determine whether a ***ex*** b, Algorithm 1 must be repeated by exchanging the roles of a and b, so as to check the case in which an occurrence of a follows an occurrence of b. If the algorithm also returns true in this case, we conclude that a ***ex*** b.

Next, we discuss the pseudocode as presented in Algorithm 1. The reasoning is analogous when the roles of a and b are exchanged.

First, the algorithm checks whether there is any maximal path in the full prefix L with both a and b. If a does not exclude b on the prefix, then it will certainly not exclude it on the whole tree. Otherwise, we compute the set *sup* of nodes following an occurrence of a and such that for each $\gamma_i \in sup$, there is no $\gamma_j < \gamma_i$ following an occurrence of a. Until *sup* is empty, the function *extract* removes an arbitrary element from *sup* and assigns it to x. For each x, we consider the set $leaves(x)$ of the leaves in L_x, and we compute the minimum element of the set $(\{\mathrm{rep}(l) : l \in \mathrm{leaves}(x) \setminus \mathrm{checked}\} \cup \{x\})$. This element, denoted by m is unique, as proved in Lemma 8.

Algorithm 1 Computing Excludes

```
procedure ex(L: full prefix, a, b ∈ T) ∈ {true, false}
  if Π_L^a ∩ Π_L^b ≠ ∅ then
    return false
  end if
  sup ← min({γ_i ∈ L : γ_i follows an occurrence of a})
  checked ← ∅
  while sup ≠ ∅ do
    x ← extract(sup)
    m ← min({rep(l) | l ∈leaves(x)\ checked} ∪ {x})
    if m ≠ x then
      sup ← sup ∪{m}
      if b ∈ L_m then
        return false
      end if
      for i ∈ sup do
        if m < i then
          sup ← sup \{i}
        end if
      end for
    end if
    checked ← checked ∪ leaves(x)
  end while
  return true
```

If the returned node m is different from x, then we check whether b occurs in L_m. If it does, a does not exclude b. Otherwise, we remove from *sup* all the nodes i such that $m < i$, and we mark all the leaves over x as checked. In the next steps we will not need to visit these leaves again. When *sup* is empty we can conclude that there cannot be any occurrence of b following an occurrence of a.

Lemma 8: Let x be a node in L, and P a set of leaves in L_x. *The minimum element of the set* $\{\mathrm{rep}(\gamma) \mid \gamma \in P\} \cup \{x\}$ *is unique.*

Proof: Let $\gamma' \in \{\mathrm{rep}(\gamma) \mid \gamma \in P\}$, then $x \leq \gamma'$ or $x > \gamma'$: for each $\gamma \in P$, $\mathrm{rep}(\gamma)$ must be in $\downarrow \gamma$, and $\downarrow \gamma \subseteq \uparrow x \cup \downarrow x$. So, either $\gamma' \in \downarrow x$, or $x \leq \gamma'$. Note that $\downarrow x$ is totally ordered, since it is the initial segment of a path. Then any unordered pair $\gamma_1, \gamma_2 \in \{\mathrm{rep}(\gamma) \mid \gamma \in P\}$ must satisfy $x \leq \gamma_1$ and $x \leq \gamma_2$. Since for any subset S, the elements of $\min(S)$ are unordered, we obtain that either $\gamma \in \min\{\mathrm{rep}(\gamma) \mid \gamma \in P\}$ is unique, or $\forall \gamma \in \min\{\mathrm{rep}(\gamma) \mid \gamma \in P\} : x \leq \gamma$, which concludes the proof. ■

Lemma 9: The procedure described in Algorithm 1 is finite.

Proof: If $\Pi_L^a \cap \Pi_L^b \neq \emptyset$ then the algorithm immediately terminates. Else, the procedure terminates when the set *sup* is empty. Let n be the number of leaves in L, then initially $n_0 = |sup| \leq n$. At every iteration, at least one element x of *sup* is removed, and possibly an element $i < x$ is added. Hence, after a general iteration $|sup| \leq n_0$. For each $x \in sup$, the number of elements $i : i < x$ is finite, therefore after a finite number of steps there is no element left that can be

added to *sup*, and $k = |sup| \leq n_0$. When this point is reached, after at most k iteration the procedure terminates. ■

Theorem 2: Let $\Sigma = (P, T, F, m_0)$ *be a free-choice Petri net,* $a, b \in T$, $A = \text{msct}(\Sigma)$, $L = \text{fp}(\Sigma)$, *and ex be the procedure of Algorithm 1. Then* $\Pi_A^a \cap \Pi_A^b = \emptyset$ *iff* $ex(L, a, b) =$ true *and* $ex(L, b, a) =$ true.

Proof: The procedure $ex(L, a, b)$ checks whether some path of L containing an occurrence of a can be extended to a path of A containing an occurrence of b, and returns false in that case. It is then required to check $ex(L, b, a)$ symmetrically, to ensure that no path of A contains both a and b.

We first assume that either $ex(L, a, b)$ or $ex(L, b, a)$ return false. If $\Pi_L^a \cap \Pi_L^b \neq \emptyset$, then $\Pi_A^a \cap \Pi_A^b \neq \emptyset$ as an immediate consequence of Lemma 5.

We assume that $\Pi_L^a \cap \Pi_L^b = \emptyset$, and we can suppose, without loss of generality, that $ex(L, a, b) =$ false, the other case being symmetric with respect to a and b. Since $\Pi_L^a \cap \Pi_L^b = \emptyset$, there must be an m such that $b \in L_m$.

At every iteration of the *while* loop, the initial segment of a path of A is considered, with at least an occurrence of a and such that its final node is isomorphic to m. It then considers all its possible extensions with segments isomorphic to those in L_m. Only if some of these extensions include b the algorithm returns false, therefore some path of A has an initial segment containing occurrences of both a and b, so $\Pi_A^a \cap \Pi_A^b \neq \emptyset$.

We now assume that $\pi \in \Pi_A^a \cap \Pi_A^b \neq \emptyset$, and that the algorithm returns true, to derive a contradiction.

If a and b occur in $\pi_{|L}$, the algorithm returns false in the first conditional statement. We consider the case in which a or b do not occur in $\pi_{|L}$. As a consequence, and by Lemma 6, there cannot be any arc in π with occurrences of both a and b. Without loss of generality, we can assume that the first occurrence of a precedes the first occurrence of b in π. We may also suppose that a occurs in $\pi_{|L}$. Otherwise, we can successively peel π until a occurs in the prefix. We also assume that there are n nodes between $\gamma_0 = \text{leaf}(\pi_{|L})$ and the first occurrence of b. Let $\pi_0 = \pi_{|L}$, and suppose that for $0 \leq j$, π_j is an initial segment of π with an occurrence of a, but none of b. Suppose that $\text{leaf}(\pi_j)$ is isomorphic to some leaf γ_j of L, and note that if it were a deadlock, we would have $\pi_j = \pi$, so b would occur in π_j. Then $\text{rep}(\gamma_j)$ must be well-defined. Furthermore, by Lemma 3, $L_{\text{rep}(\gamma_j)}$ is isomorphic to some prefix of $A_{\text{leaf}(\pi_j)}$, so we may define π_{j+1} to be the initial segment of π, obtained by extending π_j with a maximal path of $L_{\text{rep}(\gamma_j)}$. Then π_{j+1} still contains an occurrence of a, and its final node is isomorphic to some leaf γ_{j+1} of L. Next, we show that b cannot occur in π_{j+1}.

Let m be as in Algorithm 1, it follows from its definition, and Lemma 8, that $L_{\text{rep}(\gamma_j)} \subseteq L_m$. Since by assumption, the algorithm does not return false, there is no occurrence of b in L_m, and so no occurrence of b can appear in $L_{\text{rep}(\gamma_j)}$. Hence b cannot occur π_{j+1}. Let $k \geq n$, we just showed by induction that b does not occur in π_k, but it must have at least n nodes after γ_0, and since it is a prefix of π, it must have an occurrence of b. This is a contradiction, so either $\Pi_A^a \cap \Pi_A^b = \emptyset$ or the algorithm returns false. ■

In the next section, we will present a more general algorithm, that also allows for checking the excludes relation. In order to justify that Algorithm 1 is in general a better option, we here provide a means of comparing the two solutions.

During the first *if*, Algorithm 1 analyses all the arcs of the tree exactly once. In the *while* loop, the algorithm crosses each arc at most once. Indeed, an appropriate implementation of the condition $b \in L_m$ would not require to check the arcs of a sub-tree that has already been analysed. As discussed above, in order to determine whether a ***ex*** b, the *while* loop must be run twice, therefore the algorithm will check $|L|$ arcs and so the complexity of determining whether a given pair of transitions excludes each other is $O(|L|)$. In order to determine excludes for all the pairs of transitions, this analysis must be repeated for all the $|T| * |T - 1|$ pairs of distinct transitions, therefore the complexity is $O(|L| * |T|^2)$. Note that in information-flow analysis, we are only interested in excludes relation between low and high transitions, not all pairs of transitions.

It will be shown, in the next section that Algorithm 2 would require, in the same conditions, to perform $O(|L|^2 * 2^{|T|})$ visits to the nodes of L. Therefore, it is in general more convenient to use Algorithm 1 when only the excludes relation needs to be checked.

C. COMPUTING FOOTPRINTS OF MAXIMAL RUNS

So far, we have considered information flow among two chosen transitions of a Petri net. The reveals and excludes relations determine if it may be inferred from the observation of one transition whether the other occurs, or does not occur in the same maximal run. In this section, we consider extensions of these concepts to subsets of transitions.

A natural way of extending information-flow analysis among the transitions of a Petri net, when it comes to information-flow security, is to determine whether observing a set of transitions causes an unwanted information flow about another set of transitions. Extended-reveals relation models such information flow (see Definition 2).

A general approach to check extended-reveals requires to determine the footprints of all runs of the system. Given a Petri net Σ, the footprint of a maximal configuration C of $\text{unf}(\Sigma)$ is the set of transitions that can be observed through the corresponding run, i.e., $\lambda(C)$. Thanks to the bijection between maximal configurations of the unfolding and maximal paths of the msc-tree, let S be the set of labels found on the edges of a maximal path of the msc-tree, $\lambda(S)$ is the footprint of the corresponding maximal run of the system, and each such footprint can be retrieved in this way.

Furthermore, the set of footprints of all maximal paths of a msc-tree can be computed from its full prefix. This is made possible by two key features. First, any path of the msc-tree can be reconstructed solely with the information provided by its full prefix. This is enabled by the following properties:

(a) all reachable markings of a msc-tree are represented in the full prefix; (b) each maximal path of a full prefix which does not end in a deadlock ends in a node corresponding to a marking which is repeated in that path; and (c) the sub-trees of msct(Σ) whose roots correspond to the same marking are isomorphic. Second, since the set of labels of the system is finite, there are only finitely many possible footprints of runs. Then only a finite collection of maximal paths needs to be explored, making sure that each possible footprint is represented in the collection. Furthermore, only a finite prefix of each such path needs to be explored, if we make sure that it displays all labels that will eventually occur along the path. These notions are formalised in the following lemma.

Lemma 10: Let A be an msc*-tree, and π be a finite prefix of some maximal path of A. Consider the final node γ of π, and suppose there is some other node γ' in π such that $\lambda(\gamma) = \lambda(\gamma')$ and $\lambda(C(\gamma)) = \lambda(C(\gamma'))$. Then the following hold*:

a) *There is a maximal path π' of A such that $\lambda(\text{conf}(\pi')) = \lambda(C(\gamma))$, and π is a prefix of π'.*
b) *For each maximal path $\pi_\gamma \neq \pi'$ of A containing γ, there is a maximal path $\pi_{\gamma'}$ of A such that $\gamma' \in \pi_{\gamma'}, \gamma \notin \pi_{\gamma'}$, and $\lambda(\text{conf}(\pi_\gamma)) = \lambda(\text{conf}(\pi_{\gamma'}))$.*

Proof: Let $s_0 = \gamma' \uparrow \pi$, so that $\pi = (\pi \downarrow \gamma') \cdot s_0$. Then $\lambda(C(\gamma)) = \lambda(\text{conf}(\pi)) = \lambda(\text{conf}(\pi \downarrow \gamma')) \cup \lambda(\text{conf}(s_0)) = \lambda(C(\gamma')) \cup \lambda(\text{conf}(s_0))$, and since $\lambda(C(\gamma)) = \lambda(C(\gamma'))$, we derive that $\lambda(\text{conf}(s_0)) \subseteq \lambda(C(\gamma'))$.

In order to show a), suppose that for some $k \in \mathbb{N}$ there is a maximal path π_k of A, and a sequence of segments $\{s_i\}_{i \leq k}$ such that $\forall i \leq k : s_i \simeq s_0$, and $(\pi \downarrow \gamma') \cdot s_0 \cdots s_k$ is a prefix of π_k. In particular, this is true for $k = 0$. Since all the s_i are isomorphic, and can be concatenated, it must hold for each of them that its initial and final nodes γ_i, and γ_{i+1} satisfy $\lambda(\gamma_i) = \lambda(\gamma_{i+1})$. In particular, $\gamma_k \simeq \gamma_{k+1}$, and it follows from Lemma 3 that $A_{\gamma_k} \simeq A_{\gamma_{k+1}}$ (see Fig. 5). Put $\gamma_0 = \gamma'$ and $\gamma_1 = \gamma$, clearly $\forall i \leq k+1 : \lambda(\gamma_i) = \lambda(\gamma_0)$ and $A_{\gamma_i} \simeq A_{\gamma_0}$. Note that $\gamma_k \uparrow \pi_k$ is a maximal path of A_{γ_k}, and that s_k is its prefix. Then there must be a maximal path π'_k in $A_{\gamma_{k+1}}$ isomorphic to $\gamma_k \uparrow \pi_k$, with a prefix s_{k+1} isomorphic to s_k, and therefore also to s_0. Then $\pi_{k+1} = (\pi_k \downarrow \gamma_{k+1}) \cdot \pi'_k$ is a maximal path of A with prefix $(\pi \downarrow \gamma') \cdot s_0 \cdots s_{k+1}$. By induction, we may derive that there is a sequence of isomorphic segments $\{s_i\}_{i \in \mathbb{N}}$, and a path $\pi' = (\pi \downarrow \gamma') \cdot s_0 \cdots s_k \cdots$, maximal in A. Furthermore, since the segments s_k are all isomorphic, we have that $\forall k \in \mathbb{N} : \lambda(\text{conf}(s_k)) = \lambda(\text{conf}(s_0)) \subseteq \lambda(C(\gamma))$. Hence, $\lambda(\text{conf}(\pi')) = \lambda(C(\gamma))$.

This setting also allows one to show b). Consider a maximal path $\pi_\gamma \neq \pi'$ of A, containing γ. There must be some $k \in \mathbb{N}$ such that $\gamma_k \in \pi_\gamma$, but $\gamma_{k+1} \notin \pi_\gamma$. But $A_{\gamma_k} \simeq A_{\gamma'}$ implies that $\gamma_k \uparrow \pi_\gamma$ is isomorphic to some maximal path $\pi'_{\gamma'}$ of $A_{\gamma'}$. Let $\pi_{\gamma'} = (\pi_\gamma \downarrow \gamma') \cdot \pi'_{\gamma'}$. Clearly $\gamma \notin \pi_{\gamma'}$, since otherwise we would have that $\gamma_{k+1} \in \pi_\gamma$. Now note that since $\forall k \in \mathbb{N} : \lambda(\text{conf}(s_k)) = \lambda(\text{conf}(s_0)) \subseteq \lambda(C(\gamma))$, then $\lambda(\text{conf}(\pi_\gamma)) = \lambda(C(\gamma') \cup \lambda(\text{conf}(\gamma_k \uparrow \pi_\gamma))$. Since $\lambda(\text{conf}(\gamma_k \uparrow \pi_\gamma)) = \lambda(\text{conf}(\pi'_{\gamma'}))$, we have that $\lambda(\text{conf}(\pi_\gamma)) = \lambda(\text{conf}(\pi_{\gamma'}))$. ■

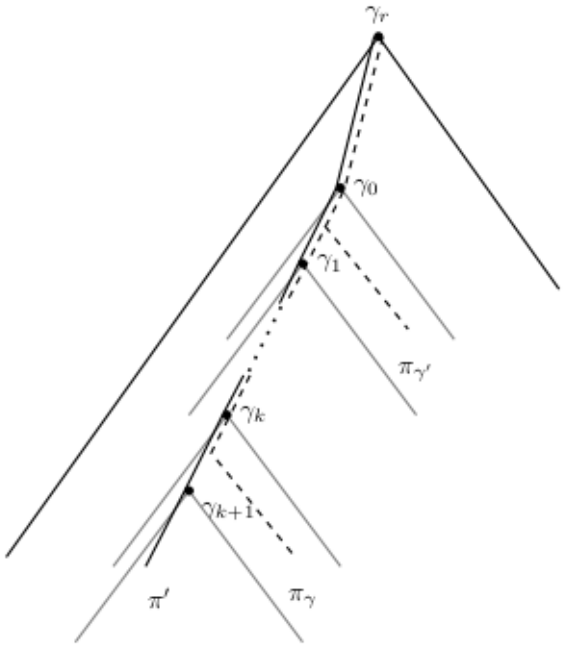


FIGURE 5. $\gamma_0 = \gamma'$, and $\gamma_1 = \gamma$. The grey sub-trees, rooted at the nodes γ_n are all isomorphic as $n \in \mathbb{N}$. The path π' contains all the nodes γ_n, and $\forall n \in \mathbb{N} : \gamma_n \uparrow \pi' \downarrow \gamma_{n+1} \simeq \gamma_0 \uparrow \pi' \downarrow \gamma_1$. For each path π_γ there is a path $\pi_{\gamma'}$ such that $\gamma_k \uparrow \pi_\gamma \simeq \gamma_0 \uparrow \pi_{\gamma'}$, and $\gamma_1 \notin \pi_{\gamma'}$.

This lemma provides a stop condition for an exploration of the msc-tree A, with the aim of finding footprints. Implication b) provides a bound to the number of maximal paths that need to be effectively visited, whereas a) lets us know when to stop exploring a path.

Algorithm 2 simulates such an exploration of A, in recursive depth first, by successively exploiting Lemma 3 to explore sub-trees of the full prefix L.

We suppose that the full prefix L is implemented thanks to the structure msct_node. Aside from the field *children*, standard in a tree structure, each node γ gathers the following information. $\gamma.m$ is the marking $\lambda(\gamma)$ corresponding to the interpretation of γ as a B-cut. Recall that each arc (γ_p, γ) of the tree is labelled with the set of events V corresponding to the step it represents. We suppose to have stored the labelling of this step $\lambda(V)$, which is a set of transitions of the net system, in the variable $\gamma.\Lambda$. When γ is a leaf of the prefix which is not a deadlock, then rep(γ) is a well-defined node of the prefix, and γ.ancestor stores a pointer to that node. Finally, we need an additional field γ.seen that is required for Algorithm 2 to terminate. It shall be initialised to $\emptyset$ and will accumulate a collection of partial footprints.

Theorem 3: Let $A = \text{msct}(\Sigma)$, L be its full prefix, and γ_r be the root of L. Let f be the function defined by Algorithm 2. Then $f(\gamma_r, \emptyset)$ returns $\{\lambda(\text{conf}(\pi)) \mid \pi \in \Pi_A\}$.

Proof: f is a recursive function that simulates a depth first exploration of A on its full prefix L. In order to formalise

Algorithm 2 Computing footprints

```
structure msct_node γ:
    m ⊆ P ..... λ(γ)
    Λ ⊆ T ..... λ(V)
    seen ⊆ 2^T ..... ∅
    ancestor: msct_node ..... rep(γ)
    children: {msct_node}
end structure

procedure f(γ: msct_node, c ⊆ T) ⊆ 2^T
    value ← ∅
    if γ.children = ∅ then
        if γ.m ∉ deadlocks(Σ) and c ∉ γ.seen then
            γ.seen ← γ.seen ∪ {c}
            for γ_i ∈ γ.ancestor.children do
                value ← value ∪ f(γ_i, c ∪ γ_i.Λ)
            end for
        else value ← {c}
        end if
    else
        for γ_i ∈ γ.children do
            value ← value ∪ f(γ_i, c ∪ γ_i.Λ)
        end for
    end if
    return value
end procedure
```

this, we will say that a pair $(\gamma', c) \in L \times 2^T$ simulates a node γ of A whenever $\lambda(\gamma) = \lambda(\gamma')$, and $c = \lambda(C(\gamma))$. Note that since $L \subseteq A$, out of Lemma 3 it holds that $A_{\gamma'} \simeq A_\gamma$ whenever (γ', c) simulates γ, independently of c.

First, we show that for each call $f(\gamma', c)$, there is a node γ simulated by (γ', c). Indeed, the initial call $f(\gamma_r, \emptyset)$ satisfies that $\lambda(C(\gamma_r)) = \emptyset$. Now suppose that we enter the body of the function from a call $f(\gamma', c)$ where (γ', c) simulates some γ of A. Further recursive calls are performed either if γ' is not a leaf of L, or when γ' is a leaf which is not a deadlock.

In the first case, it follows from $A_{\gamma'} \simeq A_\gamma$ that a node γ'_i of L is a child of γ' if, and only if γ has a child γ_i such that $\lambda(\gamma_i) = \lambda(\gamma'_i)$. Furthermore, if V and V' are the steps labelling the arcs (γ, γ_i), and (γ', γ'_i) respectively, it holds that $\lambda(V) = \lambda(V')$. It is clear that each such γ_i satisfies $\lambda(C(\gamma_i)) = \lambda(C(\gamma) \cup V)$, and it follows that $\lambda(C(\gamma_i)) = \lambda(C(\gamma)) \cup \lambda(V') = c \cup \gamma'_i.\Lambda$.

In the second case, simply note that $\mathrm{rep}(\gamma')$ is well-defined and $\lambda(\mathrm{rep}(\gamma')) = \lambda(\gamma') = \lambda(\gamma)$. Then the previous argument applies to $\mathrm{rep}(\gamma')$ instead of γ'. It follows that each maximal path of A is explored, up to simulation, and until the stop condition $c \in \gamma'.\mathrm{seen}$ holds.

Next, we show that each call $f(\gamma', c)$ returns a set containing only footprints of maximal paths of A. In fact, it returns the footprints of all maximal paths of A which contain a node γ simulated by (γ', c), unless these footprints are already returned at a previous call $f(\gamma', c)$. Note in the function body, that the only return statement concerns the variable value, which is set to $\emptyset$ at the beginning of the call. We may suppose that all recursive calls inside the body are effectively returning a set containing only valid footprints.

We distinguish two cases, either γ' is a leaf of L or not. If it is not, then the algorithm will enter a *for* loop, at each iteration of which the content of the variable value will be extended with whatever is returned by a recursive call to function f. Thus, by hypothesis, only valid footprints are added to the contents of value.

If γ' is a leaf of L, which is not a deadlock, and such that $c \notin \gamma'.\mathrm{seen}$, then the same argument applies. If $c \in \gamma'.\mathrm{seen}$, variable value gets the singleton $\{c\}$. In this case, c must have been added to $\gamma'.\mathrm{seen}$ in a previous call $f(\gamma', c)$, and so there are two nodes γ_0, and γ_1 of A, both simulated by (γ', c).

Suppose there is a maximal path of A which contains both γ_0 and γ_1, then we are in the conditions of Lemma 10. By 10.a), c must be the footprint of some maximal path of A. Note that in this case, the simulated visit to γ_1 must return before that to γ_0 does, and so the values to be returned by the latter depend on those returned by the former. However, from 10.b) we may deduce that if the footprint of a maximal path containing γ_1 is different from c, then it is returned by the first call $f(\gamma', c)$, independently from the second. Indeed, for each such path, there is a path with the same footprint containing γ_0, but not γ_1, so its footprint is computed without relying on the second call. Hence, c is the only footprint to be returned by the second call, which is not redundant with those returned by the first.

If on the contrary, there is no maximal path of A containing both γ_0 and γ_1, then by the second call, the first call $f(\gamma', c)$ must have already returned and so there must have been an intermediary call $f(\gamma', c)$ simulating a visit to a node γ_2, such that there is a maximal path of A containing both γ_0 and γ_2. Then conditions of Lemma 10 apply to γ_0 and γ_2, and by the previous argument, the first call has returned the footprints of all maximal paths of A containing γ_0. Since by Lemma 3, $A_{\gamma_0} \simeq A_{\gamma_1}$, then a maximal path of A containing γ_1 has a given footprint if and only if A has a maximal path containing γ_0 with the same footprint. Furthermore, by concatenating infinitely many isomorphic copies of the segment with initial node $\mathrm{rep}(\gamma')$ and final node γ' to the path leading to γ_1, we obtain a maximal path of A whose footprint is precisely c.

The case in which γ' is a leaf of L which is a deadlock remains. But we have seen that (γ', c) simulates some γ of A, and since $A_\gamma \simeq A_{\gamma'}$, then γ must be a deadlock as well. Hence $c = \lambda(C(\gamma))$ is the footprint of the finite maximal path ending at γ.

Finally, since L and T are finite, there are only finitely many pairs $(\gamma, c) \in L \times 2^T$. Only the first time a call $f(l, c)$ is performed, the size of the sub-tree of A whose exploration is simulated increases, and always by at most the size of the full prefix. Then if w is the number of leaves of L which are not deadlocks, the size of the explored sub-tree is at most $|L| + w * 2^{|T|} * |L|$. This value is an upper bound to the total

number of calls to f, which ensures termination and provides a worst case asymptotic complexity $O(|L|^2 * 2^{|T|})$. ■

This proof concludes that the footprints of maximal runs of a system can be computed on its full prefix in a finite number of steps. This enables us to identify all the extended-reveals relations among transitions by looking for labels in the computed footprints.

VI. EXAMPLES

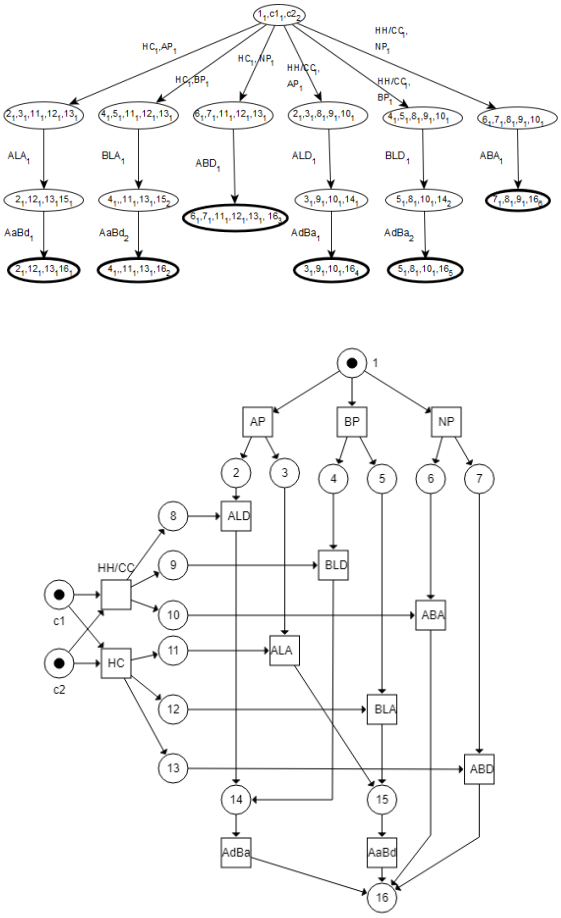
In this section we discuss some simple application scenarios in which our approach is meaningful. Our objective is twofold. On the one hand, we indicate some directions for the development of effective applications. On the other hand each scenario is chosen so as to underline a feature of the methods we have proposed. Although reveals and excludes relations are defined and studied mainly for information-flow security, we do not limit our discussions with that and we include some further discussions on the use of our methods for verification of other properties like system reliability and conformance.

A. THE DINING CRYPTOGRAPHERS

The notions of information flow we have studied in this paper allow for expressing security requirements in a very natural way. Indeed, either in systems or communication protocols, reveals and excludes relations allow one to determine whether a system is free from information flow, i.e., no information can be achieved about occurrence of the hidden actions by observing the public actions. The following example displays such a situation, in which security specifications can be checked as proposed in this work. We rely on an example developed in [40] and [41], in which the authors study a variant of the dining cryptographers protocol. Here we provide a 1-safe free-choice net modelling the protocol, and apply our methods to verify its conformance to the security specifications.

The two cryptographers Anne and Bob enjoy a meal at the restaurant. When they ask for the bill, they are informed that it has already been paid. They want to know whether one of them or their employer National Security Agency (NSA) paid, but in case one of them paid, they do not want their neighbour Eve to discover who. To this aim, they perform a protocol to exchange information in a secure way. They toss two coins, visible to both of them, and they state their parity ('agree' if the coins show the same side, 'disagree' otherwise). Eve can hear whatever they say, but she cannot see the coins. If Anne paid, she will lie about the parity of the two coins, otherwise she will tell the truth. Bob will do the same. After this procedure Anne and Bob will know who paid the bill, whereas Eve will only be able to say whether one of them or the NSA paid. But if NSA did not pay, Eve will not be able to know who paid among Anne and Bob.

This protocol is modelled with the 1-safe free choice net in Fig. 6, where the *msc*-tree of the net and the legend of the labels of each transition are also presented. Below we examine the net with our techniques.



Legend of transitions	
AP/ BP/ NP	Anne/ Bob/ NSA paid
HH/CC	the coins show two heads or two crosses
HC	one coin shows head, the other cross
ALA/ BLA	Anne/ Bob lies stating that the two coins agree
ALD/ BLD	Anne/ Bob lies stating that the two coins disagree
ABA	Anne and Bob state that the coins agree
ABD	Anne and Bob state that the coins disagree
AdBa	Anne states that the two coins disagree, whereas Bob states they agree
AaBd	Anne states that the two coins agree, whereas Bob states they disagree

FIGURE 6. The dining cryptographers net and the full prefix of its maximal step computation tree.

Example 13: The protocol must make sure (1) Eve is not able to determine who paid among Anne and Bob, (2) Anne and Bob are able to determine who paid.

The first requirement is a noninterference property. In the protocol, Eve can observe AdBa, AaBd, ABA and ABD. Everything else is hidden. We can verify the first requirement by checking whether the transitions that are observable to Eve reveal if Anne or Bob paid (AP and BP). As discussed in Section V-A, by looking at the footprints of the maximal paths of the full prefix (which is actually the whole msc-tree in this case), we see that none of the mentioned transitions reveals AP or BP, e.g., AdBa ⊬ AP, AdBa ⊬ BP, etc. We see that no combination of the observable transitions appears

in the same path, so it is not relevant to check whether any combination extended-reveals AP or BP. We can conclude that the first requirement is satisfied.

The second requirement is actually not a noninterference property but it is a property that the protocol must satisfy to ensure the required information is exchanged between Anne and Bob. This property can be expressed in terms of extended-reveals relation and can be verified by computing the footprints of the msc-*tree as discussed in Section V-C. However, in this example there are no cycles and so the full prefix is actually the whole* msc-*tree. This means further computation is not required. The set of footprints of maximal paths of the full prefix and the msc-tree are the same. The protocol is finalised by one of the following transitions: AdBa, AaBd, ABA, ABD. All are observable. Nothing else is observable to Eve, but AP is observable to Anne while BP is observable to Bob. Looking at the footprints we deduce that ABA* $\triangleright$ *NP and ABD* $\triangleright$ *NP. We also see that* $\{AdBa\} \rightarrow \{AP, BP\}$ *and* $\{AaBd\} \rightarrow \{AP, BP\}$*. This means that by observing AdBa or AaBd, the observer can deduce either Anne or Bob paid. So, while Eve is not able to understand who paid, when Bob pays, Anne understands and vice versa.*

In this example, reveals and extended-reveals relations are used to verify that the protocol is free from positive information flow and that it ensures the information exchange. Negative information flow is not relevant in this example but it is a nice exercise to look for it. The only negative information flow is the following. Occurrences of ABA and ABD imply nonoccurrence of AP and BP, i.e., ABA ***ex*** *AP, ABA* ***ex*** *BP, ABD* ***ex*** *AP and ABD* ***ex*** *BP. Since nonoccurrence of AP doesn't imply Bob paid (and similarly nonoccurrence of BP doesn't imply Anne paid), the secrets are still safe and this flow doesn't violate the security of the protocol.*

B. A BUSINESS PROCESS MODEL

Free-choice nets are proven to allow efficient analysis of various properties [14], [42]. This is achieved by eliminating a certain kind of behaviour, namely interference of choice and synchronisation. This separation comes natural for majority of process models used in practice and hence makes this particular class of Petri nets suitable for modelling business processes [12]. Here we reason about the use of our methods in the fields of process mining and business process management. We foresee that our methods can be useful in the field for tackling two major aspects: (1) information-flow security analysis by computing reveals and excludes relations, (2) conformance check by the use of footprints for characterising possible runs of the system.

We discuss our techniques on a simple example. The Petri net in Fig. 7 is adapted from a model in Chapter 2.3 of "Process Mining" book by Aalst [43]. The net models handling of compensation requests within an airline. A customer may place a request for various reasons. After a request, the ticket is checked and an inspection is performed in parallel. However, inspection can be in two different ways which is modelled via conflicting activities. Transition b and e model a critical inspection activity which is performed for suspicious or complex requests. Transition c and f model a casual inspection activity which is performed for regular requests. Decision is made only after ticket is checked and the inspection is performed. There are three possible outcomes of the decision: the requested compensation is paid, the request is declined or further processing is needed. In the first two cases the process is finalised. In the latter case the process returns to the marking $\{2, 3\}$.

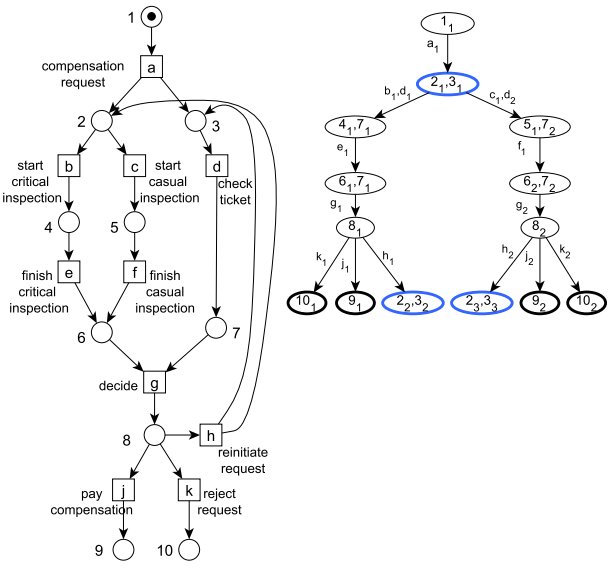


FIGURE 7. Petri net model of a business process and the full prefix of its msc-tree.

Below we apply our techniques on the net in Fig. 7.

Example 14: The full prefix of the msc-*tree of the net is included in Fig. 7. Thick black nodes indicate deadlocks whereas coloured nodes correspond to a marking which is repeated in a path. Looking at the footprints of the maximal paths of the full prefix, we can extract the reveals relations among the transitions. To start with, all the transitions reveal transition a. However, this is not a security concern. In fact, reveals relation raises an information-flow security concern only when an observable (low) transition reveals a secret (high) transition. In the example, no transition is classified as high or low but we can intuitively assume that what kind of inspection activity is performed is meant to be a secret, so transitions b, c, e and f are high transitions and assume everything else is low. We can easily deduce from the footprints these transitions are not revealed by any low transition. This means no deductions about the inspection can be done by observing the low actions. By checking reveals relations, we proved that the net is secure with respect to positive information flow. We can further analyse the net for negative information flow by checking the excludes relations with the help of Algorithm 1. In this net, no low transition excludes a high transition so we can confirm that there is no negative information flow.*

The Algorithm 2 computes footprints of all maximal paths of the msc-*tree, so the footprints of all possible maximal runs of the system, on the full prefix. From information-flow security perspective, this algorithm helps to investigate extended-reveals relation which models a more intricate flow involving more than two transitions. In this net, no low transition extended-reveals any high transition. The algorithm can also help us to characterise the possible maximal runs of the system. For example,* $\{a, b, c, d, e, f, g, h, j\}$ *is a footprint of the msc-tree of the net in Fig. 7. This means that there exists at least one maximal run of the system in which all these labels occur. On the other hand,* $\{a, b, g\}$, $\{a, c, d, f, g, j, k\}$ *or* $\{a, b, e, h\}$ *are not footprints of the maximal paths of msc-tree of the net. This means that there are no maximal runs of the system with only these labels.*

Business process management and process mining are widely studied and there is a collection of software that are effectively used in the industry (e.g., Celonis, Disco/Fluxicon) as well as scientific tools (e.g., ProM). Our theoretical results might practically contribute in the field by extending one of these software with our techniques with the aim of information-flow security analysis and/or conformance check.

C. CLIENT AND PROVIDERS

The central role of concurrency in Petri nets make them suitable models for the design of distributed workflows. They have become a standard tool for this purpose [44], and the latest industrial requirements keep motivating the use of these models.

We discuss a simple modular example, which could grow in size to express arbitrarily complex behaviours. We discuss system reliability and information-flow security in terms of reveals and excludes relations on a simple instance of this model. We model a client as a controller interacting with providers through some actions. Each action may send one or several service requests in parallel, each to a different service provider, and remains idle until all the services are performed. When all services are performed the action is concluded and control returns to the client. The client controller may implement an arbitrary program by sequentially determining which action to take.

Suppose that the client needs to perform an operation for which she requires data, and some remote computational power. She then needs to check the outcome against a data set from a different database. Fig. 8 displays a controller for such task. It accesses the database of Provider 1 by performing request $cr1$. If the query produces an error, the controller launches an emergency interrupt procedure. If the data is obtained, the firing of $cr23$ launches two requests in parallel to concurrently process the data on the remote cluster of Provider 2, and query the database of Provider 3 for the second data set. If this second query does not produce an error, then $cr23'$ may fire, allowing the system to check the processed data against this data set, possibly by invoking yet other services, and continue with the execution of its arbitrary program, and service requests.

When a data provider receives the request, it simply processes it through $pr1$ or $pr3$, which sends a positive or negative reply and enables it to reset. Provider 2 is optimised so that depending on the request, it may choose to perform the operation with algorithms $algo1$, or $algo2$.

Below, we examine the model illustrated in Fig. 8 for system reliability and information-flow security by means of reveals and excludes relations. Note that, in this setting, client can observe the transitions that are within the client component (illustrated via dashed rectangle labelled "Client"). The transitions belonging to the providers are hidden from the client. And conversely, the providers can only observe their own transitions. In the below example, the observability of the transitions is relevant for the analysis of security (noninterference) properties.

Example 15: In this model, transition $cr23'$ *reveals transitions* $s1$, $s2$, *and* $s3$, *but does not reveal transitions* $algo1$, $algo2$. *In fact, no observable transition reveals* $algo1$ *or* $algo2$. *This can be interpreted as the fact that once the client has received all the data, she is sure that the three services have been provided, but she cannot decide which algorithm was chosen by Provider 2 to perform its task. Conversely, transition* $s1$ *reveals* $cr1$, *and* $s2$ *and* $s3$ *both reveal transition* $cr23$*; i.e., the providers send a reply only when the client sends them a request. In general, reveals transition doesn't imply any order, i.e., the revealed transition may occur before or after the revealing transition or even simultaneously with it. However, knowing the structure of the net, we can safely say that the mentioned reveals relations guarantee that the services are provided only when a request is made.*

Note that $cr1$ *does not reveal* $s1$, *since the request might lead to* $error1$. *Neither does* $cr23$ *reveal* $s3$, *but it does instead reveal* $s2$ *since Provider 2 always processes the data it receives. Also,* $error1$ *and* $error3$ *exclude each other, because the system stalls after interrupt*.

These properties can easily be verified on the full prefix of msc-*tree. Note that for all practical purpose, the undefined process in Fig. 8 may be abstracted as a single transition* t. *This transition represents the set of transitions* T *which represents all the transitions in the abstracted module. In this context, for a transition* $t' \notin T$, $t \rhd t'$ *can be interpreted as* $\exists t_0 \in T : t_0 \rhd t'$, *and conversely,* $t' \rhd t$ *as* $\exists t_0 \in T : t' \rhd t_0$.

With this abstraction, the full prefix of the msc-*tree has five maximal paths. Three of these correspond to executions that end in a deadlock after the firing of interrupt, and have for respective footprints the sets* $\{cr1, pr1, error1, reset1, interrupt\}$, $\{cr1, pr1, reset1, s1, cr1', cr23, pr2, algo1, s2, cr23, pr3, error3, reset3, interrupt\}$, *and* $\{cr1, pr1, reset1, s1, cr1', cr23, pr2, algo2, s2, cr23, pr3, error3, reset3, interrupt\}$. *Note that the last two footprints only differ in the choice of Provider 2 for the algorithm. This is also the case for the two remaining paths: each depicts all transitions in the system but for*

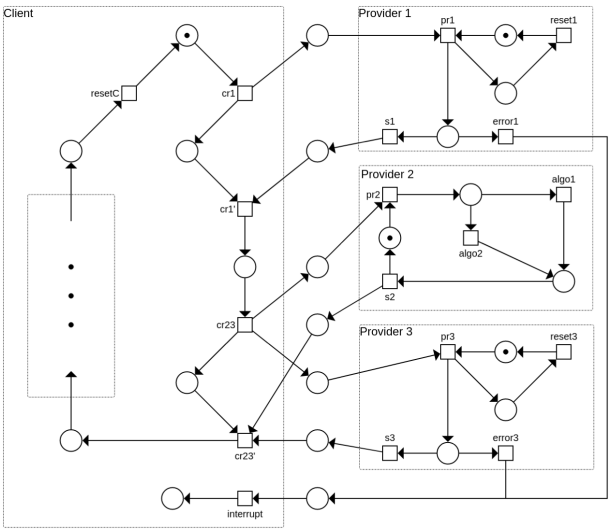


FIGURE 8. A client exchanging information with three providers.

*error*1, *error*3, *and interrupt in its footprint, and present exactly one of the labels among algo*1, *and algo*2.

*We may conclude that it is impossible, by observing the client, or the database providers, to determine whether Provider 2 used algorithm 1 or 2. On the other hand, the client may determine that Provider 1 did provide the requested service s*1, *by observing cr*1′, *cr*23, *cr*23′, *or resetC. With analogous computations all the reveals relation can be derived, as for instance cr*23 ▷ *s*1, *but cr*23 ▷̸ *algo*1.

*Another property that we may want to check is the possibility of error. It is easy to see that error*1 ***ex*** *error*3, *i.e., only one error can occur in the system that performs an emergency stop. However, by applying Algorithm 1 we can see that cr*23′ *does not exclude error*1 *(or error*3*), i.e., even when the three services were successfully provided once to the client, an error could still occur after the first resetC.*

This model can be straightforwardly extended by adding an arbitrary number of actions and providers with possibly different features. The class of properties discussed in this example can be verified on these larger models with the same techniques. In order to do so, however, some care has to be taken in modelling the system as a free-choice net.

VII. CONCLUSION

In this paper, we have introduced maximal-step computation tree which is a tree structure to represent the behaviour of *a* 1-safe free-choice Petri net under maximal-step semantics. Nodes of the tree correspond to B-cuts of the net's unfolding and arcs are labelled by the E-cuts corresponding the maximal steps of the net. We have defined a finite prefix of the tree, called full prefix. This prefix is a compact representation of a Petri net's behaviour from which all the possible maximal configurations of the corresponding unfolding can be computed.

We have shown that the full prefix forms an adequate basis for information-flow analysis of concurrent systems that can be modelled by 1-safe free-choice Petri nets. On the full prefix, we have studied reveals and excludes relations which are defined to model information flow among the transitions of a Petri net. We have proved that reveals relation can be computed on the full prefix without further computation. This can be done by traversing the tree and checking for labels. We have shown that the excludes relation can be computed on the full prefix in a finite number of steps with Algorithm 1. As a consequence, reveals and excludes based noninterference properties can be verified on the full prefix of a Petri net model.

We have presented the notion of footprint. The footprint of a path of a msc-tree is the finite set of labels of the transitions that are fired throughout the corresponding system run. A footprint can be seen as a summary of a system run independent from how many times a certain transition fired. Footprints relate to reveals and excludes relations in the following way. Transition *a* reveals transition *b* if, and only if, each maximal path of the full prefix whose footprint displays *a* also displays *b*. Transition *a* excludes transition *b* (and so *b* excludes *a*) if, and only if, no maximal path of the msc-tree has both *a* and *b* in its footprint. The extended-reveals relation can be extracted from the footprints of maximal runs of a Petri net. Algorithm 2 computes these sets when given the full prefix of the Petri net.

Although the notions of reveals, excludes and footprints are introduced and studied from the information-flow security perspective, they can be used for verification of different properties of systems that can be modelled via 1-safe free-choice nets. Our methods show potential to be used in various application areas including fault diagnosis, protocol verification, workflow verification, business process analysis.

In the near future, we plan to continue to explore the notions of msc-tree, full prefix and footprints in more details and improve our methods. We will work on extending our results to more general classes of Petri nets which will broaden the application domain. We will explore the practical use of our analysis methods on complex systems such as critical infrastructures, IoT, cloud systems and blockchains.

ACKNOWLEDGMENT

The authors would like to thank Lucia Pomello, Luca Bernardinello, and Carlo Ferigato for the fruitful discussions, and Heiko Mantel for his useful comments.

REFERENCES

[1] C. A. Petri, "Kommunikation mit automaten," Ph.D. dissertation, Dept. Sci., Tech. Univ. Darmstadt, Darmstadt, Germany, 1962.
[2] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[3] J. L. Peterson, "Petri nets," *ACM Comput. Surv.*, vol. 9, no. 3, pp. 223–252, 1977.
[4] J. Desel and G. Juhás, "What is a Petri net?" in *Unifying Petri Nets, Advances in Petri Nets* (Lecture Notes in Computer Science), vol. 2128, H. Ehrig, G. Juhás, J. Padberg, and G. Rozenberg, Eds. Berlin, Germany: Springer, 2001, pp. 1–25.
[5] A. Giua and M. Silva, "Petri nets and automatic control: A historical perspective," *Annu. Rev. Control*, vol. 45, pp. 223–239, Jun. 2018.

[6] J. Esparza and M. Nielsen, "Decidability issues for Petri nets—A survey," *J. Inf. Process. Cybern.*, vol. 30, no. 3, pp. 143–160, 1994.
[7] M. H. T. Hack, "Analysis of production schemata by Petri nets," Massachusetts Inst. Tech., Cambridge, MA, USA, Tech. Rep. MAC TR-94, 1972.
[8] E. Best, "Structure theory of Petri nets: The free choice hiatus," in *Petri Nets: Central Models and Their Properties, Advances in Petri Nets* (Lecture Notes in Computer Science), vol. 254. Bad Honnef, Germany: Springer, 1986, pp. 168–205.
[9] J. Desel and J. Esparza, *Free Choice Petri Nets*. Cambridge, U.K.: Cambridge Univ. Press, 1995.
[10] E. Smith, "On the border of causality: Contact and confusion," *Theor. Comput. Sci.*, vol. 153, nos. 1–2, pp. 245–270, 1996.
[11] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004.
[12] W. M. P. van der Aalst, "Using free-choice nets for process mining and business process management," in *Proc. 16th FedCSIS*, M. Ganzha, L. A. Maciaszek, M. Paprzycki, and D. Slezak, Eds., 2021, pp. 9–15.
[13] W. M. P. van der Aalst, "Free-choice nets with home clusters are lucent," *Fundam. Inf.*, vol. 181, no. 4, pp. 273–302, 2021.
[14] W. M. P. van der Aalst, "Reduction using induced subnets to systematically prove properties for free-choice nets," in *Proc. PETRI NETS*, vol. 12734, D. Buchs and J. Carmona, Eds., Paris, France. Cham, Switzerland: Springer, 2021, pp. 208–229.
[15] J. Engelfriet, "Branching processes of Petri nets," *Acta Inf.*, vol. 28, no. 6, pp. 575–591, 1991.
[16] J. Esparza, S. Romer, and W. Vogler, "An improvement of McMillan's unfolding algorithm," in *Formal Methods in System Design*. Heidelberg, Germany: Springer-Verlag, 1996, pp. 87–106.
[17] V. Khomenko, M. Koutny, and W. Vogler, "Canonical prefixes of Petri net unfoldings," *Acta Inf.*, vol. 40, no. 2, pp. 95–118, 2003.
[18] R. Janicki, P. E. Lauer, M. Koutny, and R. Devillers, "Concurrent and maximally concurrent evolution of nonsequential systems," *Theor. Comput. Sci.*, vol. 43, pp. 213–238, Dec. 1986.
[19] H. Mantel, "Information flow and noninterference," in *Encyclopedia Cryptography Security*, H. C. A. van Tilborg and S. Jajodia, Eds. Boston, MA, USA: Springer, 2011, pp. 605–607.
[20] J. A. Goguen and J. Meseguer, "Security policies and security models," in *Proc. IEEE Symp. Secur. Privacy*, Oct. 1982, pp. 11–20.
[21] D. Sutherland, "A model of information," in *Proc. 9th Nat. Comput. Sec. Conf.*, vol. 247, 1986, pp. 175–183.
[22] D. McCullough, "Specifications for multi-level security and a hook-up," in *Proc. IEEE Symp. Secur. Privacy*, Apr. 1987, p. 161.
[23] R. Focardi and R. Gorrieri, "A taxonomy of security properties for process algebras," *J. Comput. Secur.*, vol. 3, no. 1, pp. 5–34, 1995.
[24] J. McLean, "A general theory of composition for a class of 'possibilistic' properties," *IEEE Trans. Softw. Eng.*, vol. 22, no. 1, pp. 53–67, Dec. 1996.
[25] H. Mantel, "Possibilistic definitions of security—An assembly kit," in *Proc. 13th CSFW*, Cambridge, U.K., 2000, pp. 185–199.
[26] H. Mantel, "A uniform framework for the formal specification and verification of information flow security," Ph.D. dissertation, Dept. Comput. Sci., Saarland Univ., Saarbrücken, Germany, 2003.
[27] H. Mantel, "The framework of selective interleaving functions and the modular assembly kit," in *Proc. FMSE*, Fairfax, VA, USA, V. Atluri, P. Samarati, R. Küsters, and J. C. Mitchell, Eds., 2005, pp. 53–62.
[28] N. Busi and R. Gorrieri, "A survey on non-interference with Petri nets," in *Petri Nets, Advances*, vol. 3098. Berlin, Germany: Springer-Verlag, 2003, pp. 328–344.
[29] N. Busi and R. Gorrieri, "Structural non-interference in elementary and trace nets," *Math. Struct. Comput. Sci.*, vol. 19, no. 6, pp. 1065–1090, 2009.
[30] S. Frau, R. Gorrieri, and C. Ferigato, "Petri net security checker: Structural non-interference at work," in *Proc. 5th FAST*, vol. 5491, P. Degano, J. D. Guttman, and F. Martinelli, Eds., Malaga, Spain. Berlin, Germany: Springer-Verlag, 2008, pp. 210–225.
[31] E. Best, P. Darondeau, and R. Gorrieri, "On the decidability of non interference over unbounded Petri nets," in *Proc. 8th SecCo*, vol. 51, K. Chatzikokolakis and V. Cortier, Eds., Paris, France, 2010, pp. 16–33.
[32] P. Baldan and A. Carraro, "Non-interference by unfolding," in *Proc. PETRI NETS*, vol. 8489, G. Ciardo and E. Kindler, Eds., Tunis, Tunisia. Berlin, Germany: Springer-Verlag, 2014, pp. 190–209.
[33] P. Baldan and A. Beggiato, "Multilevel transitive and intransitive non-interference, causally," *Theor. Comput. Sci.*, vol. 706, pp. 54–82, Dec. 2018.
[34] F. Basile, G. D. Tommasi, and C. Sterle, "Noninterference enforcement via supervisory control in bounded Petri nets," *IEEE Trans. Autom. Control*, vol. 66, no. 8, pp. 3653–3666, Sep. 2021.
[35] G. Kılınç, "Formal notions of non-interference and liveness for distributed systems," Ph.D. dissertation, Dept. Inform., Syst. Commun., Univ. Milano-Bicocca, DISCo, Milano, Italy, 2016.
[36] L. Bernardinello, G. Kılınç, and L. Pomello, "Non-interference notions based on reveals and excludes relations for Petri nets," *ToPNoC*, vol. 11, pp. 49–70, Mar. 2016.
[37] S. Haar, "Unfold and cover: Qualitative diagnosability for Petri nets," in *Proc. 46th IEEE Conf. Decis. Control*, Dec. 2007, pp. 1886–1861.
[38] S. Haar, C. Rodríguez, and S. Schwoon, "Reveal your faults: It's only fair!" in *Proc. 13th ACSD*, Barcelona, Spain, 2013, pp. 120–129.
[39] S. Balaguer, T. Chatain, and S. Haar, "Building tight occurrence nets from reveals relations," in *Proc. 11th ACSD*, B. Caillaud, J. Carmona, and K. Hiraishi, Eds., Newcastle Upon Tyne, U.K., 2011, pp. 44–53.
[40] J. W. Bryans, M. Koutny, and P. Y. A. Ryan, "Modelling opacity using Petri nets," *Electron. Notes Theor. Comput. Sci.*, vol. 121, pp. 101–115, Feb. 2005.
[41] L. Mazaré, "Using unification for opacity properties," in *Proc. WITS*, Barcelona, Spain, 2004, pp. 165–176.
[42] J. Esparza, "Advances in quantitative analysis of free-choice workflow Petri nets (invited talk)," in *Proc. 24th TIME*, vol. 90, S. Schewe, T. Schneider, and J. Wijsen, Eds. Dagstuhl, Germany: Schloss Dagstuhl, 2017, pp. 1–6.
[43] W. M. P. van der Aalst, *Process Mining—Data Science Action*. Berlin, Germany: Springer, 2016.
[44] W. M. P. van der Aalst, "The application of Petri nets to workflow management," *J. Circuits Syst. Comput.*, vol. 8, pp. 21–66, Feb. 1998.

FEDERICA ADOBBATI received the Graduate degree in computer science from the University of Milano-Bicocca, where she is currently pursuing the Ph.D. degree in computer science. Her research interest includes the formal methods for the verification of properties in multi-agent concurrent systems.

GÖRKEM KILINÇ SOYLU received the B.S. degree in computer engineering from the Izmir Institute of Technology, in 2009, the M.S. degree in computer engineering from Izmir Yaşar University, in 2012, and the Ph.D. degree in computer science from the University of Milano-Bicocca, in 2016. She had continued her research with the University of Milano-Bicocca as a Postdoctoral Researcher, until she joined the MAIS Team, TU Darmstadt, in November 2017. Her research interests include formal models of concurrent systems and information-flow security.

ADRIÁN PUERTO AUBEL received the Graduate degree in pure mathematics from the Complutense University of Madrid, and the Ph.D. degree in computer science from the University of Milano-Bicocca. Since then, he has undertaken researcher positions at Inria and in the private sector. He is currently a Researcher with the University of Groningen. His research interests include the formal models of distributed systems and logic.