Newcastle University

COMPUTING SCIENCE

Unfolding CSPT-nets

Bowen Li and Maciej Koutny

TECHNICAL REPORT SERIES

No. CS-TR-1463 **April 2015**

TECHNICAL REPORT SERIES

No. CS-TR-1463 **April, 2015**

Unfolding CSPT-nets

B. Li and M. Koutny

Abstract

Communication structured occurrence nets CSONs are the basic variant of structured occurrence nets which have been introduced to characterise the behaviours of complex evolving systems. A CSON has the capability of portraying different types of interaction between systems by using special elements to link with multiple (component) occurrence nets. Communication structured place transition nets CSPT-nets are the system-level counterpart of CSONs.

In this paper, we investigate CSPT-nets unfoldings containing representations of all the single runs of the original nets captured by CSONs. We develop several useful notions related to CSPT-net unfoldings, and then present an algorithm for constructing the new class of unfolding.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

LI, B; KOUTNY, M;
Unfolding CSPT-nets
[By] B. Li and M. Koutny
Newcastle upon Tyne: Newcastle University: Computing Science, 2015.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1463)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1463

Abstract

Communication structured occurrence nets CSONs are the basic variant of structured occurrence nets which have been introduced to characterise the behaviours of complex evolving systems. A CSON has the capability of portraying different types of interaction between systems by using special elements to link with multiple (component) occurrence nets. Communication structured place transition nets CSPT-nets are the system-level counterpart of CSONs.

In this paper, we investigate CSPT-nets unfoldings containing representations of all the single runs of the original nets captured by CSONs. We develop several useful notions related to CSPT-net unfoldings, and then present an algorithm for constructing the new class of unfolding.
.

About the authors

Maciej Koutny is a Professor in the School of Computing Science, Newcastle University. His research interests centre on the theory of distributed and concurrent systems, including both theoretical aspects of their semantics and application of formal techniques to the modelling and verification of such systems; in particular, model checking based on net unfoldings. He has also investigated non-interleaving semantics of priority systems, and the relationship between temporal logic and process algebras. Recently, he has been working on the development of a formal model combining Petri nets and process algebras as well as on Petri net based behavioural models of membrane systems.

Bowen is working on the EPSRC funded project UNCOVER (UNderstanding COmplex system eVolution through structurEd behaviours). An overall goal of UNCOVER is to develop a rigorous methodology supported by a toolkit based on structured occurrence nets, in order to provide an effective approach to acquiring and exploiting behavioural knowledge of a complex evolving system.

Suggested keywords

PETRI NETS
STRUCTURED OCCURRENCE NETS
NET UNFOLDING
CONCURRENCY

Unfolding CSPT-nets

Bowen Li and Maciej Koutny

School of Computing Science, Newcastle University
Newcastle upon Tyne NE1 7RU, United Kingdom
{bowen.li,maciej.koutny}@ncl.ac.uk

Abstract. Communication structured occurrence nets (CSONs) are the basic variant of structured occurrence nets which have been introduced to characterise the behaviours of complex evolving systems. A CSON has the capability of portraying different types of interaction between systems by using special elements to link with multiple (component) occurrence nets. Communication structured place transition nets (CSPT-nets) are the system-level counterpart of CSONs. In this paper, we investigate CSPT-nets unfoldings containing representations of all the single runs of the original nets captured by CSONs. We develop several useful notions related to CSPT-net unfoldings, and then present an algorithm for constructing the new class of unfolding.

1 Introduction

A complex evolving system consists of a large number of sub-systems which may proceed concurrently and interact with each other or with the external environment while its behaviour is subject to modification by other systems. A typical example would be a federated cloud [15] combining multiple interacting cloud providers, with each cloud being composed of a huge amount of services. The cloud infrastructure can suffer from component break-downs, reconfigurations and replacement, and the software is continually updated or patched. The communication between sub-systems may either be asynchronous or synchronous.

Structured occurrence nets (SONs) [8, 13, 14] are a Petri net based formalism that can be used to model the behaviours of complex evolving system. The concept extends that of occurrence nets [1] which are directed acyclic graphs that represent causality and concurrency information concerning a single execution of a system. In SON, multiple related occurrence nets are combined by means of various formal relationships; in particular, in order to express dependencies between interacting systems. Communication structure occurrence nets (CSONs) are the basic variant of SONs. The model has the capability of portraying different types of interaction between systems. A CSON involves occurrence nets that are connected by channel places representing synchronous or asynchronous communications. [7] introduced a system-level counterpart of CSONs called communication structured place transition nets (CSPT-nets). The nets are built out of the place/transition nets (PT-nets), which are connected by channel places allowing both synchronous and asynchronous communication.

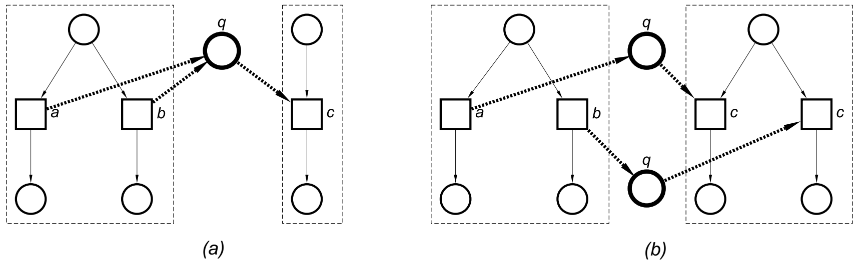

Fig. 1. (a)A CSPT-net example; and (b) its unfolding using standard unfolding approach.

The standard Petri nets unfoldings, introduced in [2, 11], are a technique supporting effective verification of concurrent systems modeled by Petri nets (throughout this paper, Petri net related concepts, such as configuration, unfolding, merged process, will be referred to as *standard*). The method relies on the concept of net unfolding which can be seen as the partial order behaviour of a concurrent system. The unfolding (or branching process) of a net is usually infinite, but for bounded Petri nets one can construct a finite complete prefix of the unfolding containing enough information to analyse important behavioural properties. [9] investigated branching processes of CSPT-nets (CSPT-net unfoldings). As in the standard net theory, CSPT branching processes act as a 'bridge' between CSPT-nets and their processes captured by CSONs (i.e., the branching processes of a CSPT-net contains a representation of all the possible single runs of the original net). In order to reduce the complexity of branching processes of CSPT-nets, we adapt the notion of occurrence depth which was originally developed for merged processes [5]. Figure 1(a) shows a CSPT-net with (asynchronous) communication between the two component PT-nets. In this case, the unfolding of is isomorphic to the original CSPT-net. Note that in the standard unfolding approach there would be a duplicated event c as the additional consumer for the input place q (figure 1(b)).

In this paper, we introduce and discuss several key properties of branching processes of CSPT-nets. We also present an algorithm for constructing CSPT-net unfoldings, generalising the unfolding algorithm introduced in [9] which could only handle channel places with a single input and a single output transition. In particular, the new algorithm takes into account the occurrence depth of events, and fusees nodes which have same behaviours during the unfolding. In this way, the size of the resulting net can be significantly reduced when compared with the standard unfolding approach.

The paper is organised as follows. Section 2 provides basic notions concerning Petri nets and their unfoldings. Section 3 presents the main concepts of communication structured net theory, including CSON-nets, CSPT-nets and CSPT branching processes. In section 4, we discuss finite complete prefixes of CSPT branching processes and related properties. The CSPT unfolding algorithm is provided in Section 5. Section 6 discusses future works and concludes the paper.

2 Basic Definitions

We assume that the reader is familiar with the basic notions concerning Petri nets and their unfoldings, which can be found in, e.g., [1,2,11]. Throughout the paper, a *multiset* over a set X is a function $\mu : X \to \mathbb{N}$, where $\mathbb{N} = \{0, 1, 2, \ldots\}$. A multiset may be represented by explicitly listing its elements with repetitions. For example $\{a, a, b\}$ denotes the multiset such that $\mu(a) = 2$, $\mu(b) = 1$ and $\mu(x) = 0$ for $x \in X \backslash \{a, b\}$.

2.1 PT-nets

A *net* is a triple $N = (P, T, F)$ such that P and T are disjoint sets of respectively *places* and *transitions* (collectively referred to as *nodes*), and $F \subseteq (P \times T) \cup (T \times P)$ is the *flow* relation. The *inputs* and *outputs* of a node x are defined as ${}^\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$. Moreover, ${}^\bullet x^\bullet = {}^\bullet x \cup x^\bullet$. It is assumed that the inputs and outputs of a transition are nonempty sets.

Two nodes, x and x', are in *conflict* if there are distinct transitions, t and t', such that ${}^\bullet t \cap {}^\bullet t' \neq \varnothing$ and $(t, x) \in F^*$ and $(t', x') \in F^*$. We denote this by $x \# x'$. A node x is in *self-conflict* if $x \# x$.

A *place transition net* (PT-net) is a tuple $PT = (P, T, F, M_0)$, where (P, T, F) is a finite net, and $M_0 : P \to \mathbb{N}$ is the *initial marking* (in general, a marking is a multiset of places).

A *step* U is a non-empty multiset of transitions of PT. It is *enabled* at a marking M if

$$M(p) \geq \sum_{t \in p^\bullet} U(t) \ ,$$

for every place p. In such a case, the *execution* of U leads to a new marking M' given by

$$M'(p) = M(p) + \sum_{t \in {}^\bullet p} U(t) - \sum_{t \in p^\bullet} U(t) \ ,$$

for every $p \in P$. We denote this by $M[U\rangle M'$. A *step sequence* of PT is a sequence $\lambda = U_1 \ldots U_n$ $(n \geq 0)$ of steps such that there exist markings $M_1, \ldots, M_n$ satisfying:

$$M_0[U_1\rangle M_1, \ldots, M_{n-1}[U_n\rangle M_n$$

The *reachable markings* of PT are defined as the smallest (w.r.t. $\subseteq$) set *reach*(PT) containing M_0 and such that if there is a marking $M \in$ *reach*(PT) and $M[U\rangle M'$, for a step U and a marking M', then $M' \in$ *reach*(PT).

PT is *k-bounded* if, for every reachable marking M and every place $p \in P$, $M \leq k$, and *safe* if it is 1-bounded. The markings of a safe PT-net can be treated as sets of places.

2.2 Branching processes of PT-nets

A net $ON = (P, T, F)$, with places and transitions called respectively *conditions* and *events*, is a *branching occurrence net* if the following hold:

- F is acyclic and no transition $t \in T$ is in self-conflict;
- $|{}^\bullet p| \leq 1$, for all $p \in P$; and
- for every node x, there are finitely many y such that $(y, x) \in F^*$.

The set of all places p with no inputs (i.e., ${}^\bullet p = \varnothing$) is the default initial state of ON, denoted by M_{ON}. In general, a *state* is any set of places.

If $|p^\bullet| \leq 1$, for all $p \in P$, then ON is a *non-branching* occurrence net. Note that in a branching occurrence net, two paths outgoing from a place will never meet again by coming to the same place (the inputs of places are at most singleton sets) nor the same transition (transitions cannot be in self-conflict).

A *branching process* of a PT-net $PT = (P, T, F, M_0)$ is a pair $\Pi = (ON, h)$, where $ON = (P', T', F')$ is a branching occurrence net and $h : P' \cup T' \to P \cup T$ is a mapping, such that the following hold:

- $h(P') \subseteq P$ and $h(T') \subseteq T$;
- for every $e \in T'$, the restriction of h to ${}^\bullet e$ is a bijection between ${}^\bullet e$ and ${}^\bullet h(e)$, and similarly for $e^\bullet$ and $h(e)^\bullet$;
- the restriction of h to M_{ON} is a bijection between M_{ON} and M_0; and
- for all $e, f \in T'$, if ${}^\bullet e = {}^\bullet f$ and $h(e) = h(f)$ then $e = f$.

There exists a maximal branching process Π_{PT}, called the *unfolding* of PT [2].

2.3 Configurations and cuts of a branching process

Let $\Pi = (ON, h)$ be a branching process of a PT-net PT, and $ON = (P', T', F')$.

A *configuration* of Π is a set of events $C \subseteq T'$ such that $\neg(e \# e')$, for all $e, e' \in C$, and $(e', e) \in F'^+ \implies e' \in C$, for every $e \in C$. In particular, the *local configuration* of an event e, denoted by $[e]$, is the set of all the events e' such that $(e', e) \in F'^*$. The notion of a configuration captures the idea of a possible history of a concurrent system, where all events must be conflict-free, and all the predecessors of a given event must have occurred.

A *co-set* of Π is a set of conditions $B \subseteq P'$ such that, for all distinct $b, b' \in B$, $(b, b') \notin F'^+$. Moreover, a *cut* of Π is any maximal (w.r.t. $\subseteq$) co-set B.

Finite configurations and cuts of Π are closely related:

- if C is a finite configuration of Π, then $Cut(C) = (M_{ON} \cup C^\bullet) \setminus {}^\bullet C$ is a cut and $Mark(C) = h(Cut(C))$ is a reachable marking of PT; and
- if M is a reachable marking of PT, then there is a finite configuration C of Π_{PT} such that $Mark(C) = M$.

Hence every marking represented in the unfolding Π_{PT} is reachable in PT, and every reachable marking of PT is represented in Π_{PT}.

3 Structuring PT-nets

In this section we recall the formal definitions concerning communication structured nets theory, including CSON-nets and CSPT-nets. We then introduce the notion of branching processes of CSPT-nets and several related properties.

The new models are able to portray different kinds of communication between separate systems. One can envisage that if a given PT-net attempts to represent several interacted systems, it will be beneficial to split the model into a set of component nets, and create specific devices to represent any communication between the subsystems. In the model we are interested in communication can be synchronous or asynchronous. Usually, the former implies that a sender waits for an acknowledgment of a message before proceeding, while in the latter the sender proceeds without waiting.

A communication structured net is composed of a set of component nets representing separate subsystems. When it is determined that there is a potential for an interaction between subsystems, asynchronous or synchronous communication link can be made between transitions (or events) in the different nets via a special element called a *channel place*. Two transitions (events) involved in a synchronous communication link must be executed simultaneously. On the other hand, transitions (events) involved in an asynchronous communication may be executed simultaneously, or one after the other.

Similarly as in the case of PT-nets, non-branching processes CSON-nets will represent single runs of CSPT-nets, while branching processes will capture full execution information of the corresponding CSPT-nets.

3.1 CSPT-nets

By generalising the definition of [7], we first introduce an extension of PT-nets which combines several such nets into one model using channel places.

Definition 1 (CSPT-net). *A* communication structured place transition net *(or* CSPT*-net) is a tuple*

$$CSPT = (PT_1, \ldots, PT_k, Q, W, M_0) \quad (k \geq 1)$$

such that each $PT_i = (P_i, T_i, F_i, M_i)$ is a safe (component) PT*-net; Q is a finite set of* channel *places; $M_0 : Q \to \mathbb{N}$ is the initial marking of the channel places; and $W \subseteq (T \times Q) \cup (Q \times T)$, where $T = \bigcup T_i$, is the flow relation for the channel places.*

It is assumed that the following are satisfied:

1. *The PT_i's and Q are pairwise disjoint.*
2. *For every channel place $q \in Q$,*
   - *the sets of inputs and outputs of q,*

$$^\bullet q = \{t \in T \mid (t, q) \in W\} \quad \text{and} \quad q^\bullet = \{t \in T \mid (q, t) \in W\} \ ,$$

   are both nonempty and, for some $i \neq j$, $^\bullet q \subseteq T_i$ and $q^\bullet \subseteq T_j$; and

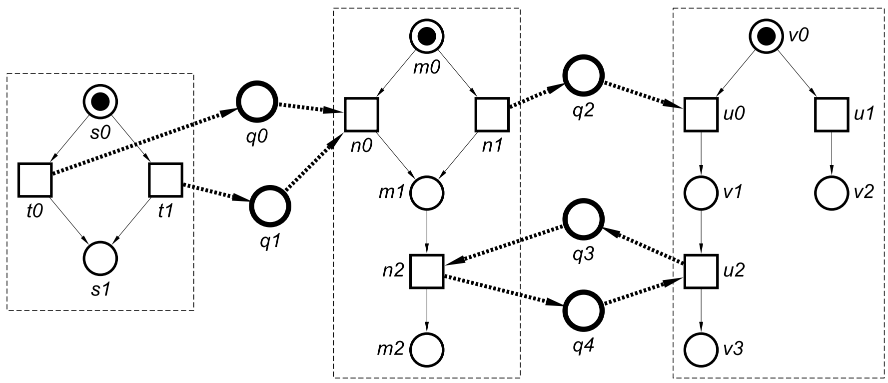

Fig. 2. A CSPT-net with three component PT-nets.

– *if* $^\bullet q^\bullet \cap T_i \neq \varnothing$ *then there is no reachable marking of* PT_i *which enables a step comprising two distinct transitions in* $^\bullet q^\bullet$. ◇

The initial marking M_{CSPT} of $CSPT$ is the multiset sum of the M_i's ($i = 0, 1, \ldots, k$), and a marking is in general a multiset of places, including the channel places.

To simplify the presentation, in the rest of this paper we will assume that the channel places in the initial states of CSPT*-nets are* empty*.*

The execution semantics of $CSPT$ is defined as for a PT-net except that a step of transitions U is *enabled* at a marking M if, for every non-channel place p,

$$M(p) \geq \sum_{t \in p^\bullet} U(t) \ ,$$

and, for every channel place q,

$$M(q) + \sum_{t \in {}^\bullet q} U(t) \geq \sum_{t \in q^\bullet} U(t) \ . \qquad (*)$$

The condition ($*$) for step enabledness caters for synchronous behaviour as step U can use not only the tokens that are already available in channel places at marking M, but also can use the tokens deposited there by transitions from U during the execution of U. In this way, transitions from U can 'help' each other individually and synchronously pass resources (tokens) among themselves. Thus, in contrast to the step execution of a PT-net where a step consists of a number of enabled transitions, the execution of a step in a CSPT-net (i.e., $M[U\rangle M'$) may involve synchronous communications (or interactions), where transitions execute simultaneously and behave as a transaction. Such a mode of execution is strictly more expressive than that used in PT-nets.

Figure 2 shows a CSPT-net which consists of three component PT-nets connected by a set of channel places (represented by circles with thick edges). To improve readability, the thick dashed lines indicate the flow relation W. Transitions n_2 and u_2 are connected by a pair of empty channel places, q_3 and q_4, forming a cycle. This indicates that these two transitions can only be executed synchronously. They will be filled and emptied synchronously when both n_2 and u_2 participate in an enabled step. On the other hand, the execution of transitions n_1 and u_0 can be either asynchronous (n_1 occurs before u_0), or synchronous (both of them occur simultaneously). A possible step sequence of Figure 2 is $\lambda = \{t_0, n_1\}\{u_0\}\{n_2, u_2\}$, where n_1 and u_0 perform an asynchronous communication. Another step sequence $\lambda' = \{t_0\}\{n_1, u_0\}\{n_2, u_2\}$ shows that n_1 and u_0 can be also executed synchronously.

Definition 1(2) means that the occurrences of transitions in ${}^\bullet q$ (as well as those in $q^\bullet$) are totally ordered in any execution of the corresponding component net PT_i. In other words, we assume that both the output access and the input access to the channel places is *sequential*. This will allow us to introduce the notion of depth at which an event which accessed a channel place has occurred.

Given a branching process derived for a component PT-net of a CSPT-net, consider an event e such that its corresponding transition is an input (or output) of a channel place q in the CSPT-net. Then the occurrence depth of such event w.r.t., the channel place q is the number of events such that they all causally precede e and their corresponding transitions are also inputs (or outputs) of the channel place q. Since the tokens flowing through channel places are based on the FIFO policy. The occurrence depth intuitively represents the number of tokens which have entered (or left) the channel place q before the occurrence of e.

Definition 2 (occurrence depth). *Let $CSPT$ be as in Definition 1, and $q \in Q$ and PT_i be such that ${}^\bullet q^\bullet \cap T_i \neq \varnothing$. Moreover, let $\Pi = (ON, h)$ be a branching process of PT_i, and e be an event of $ON = (P', T', F')$ such that $h(e) \in {}^\bullet q^\bullet$.*

The depth *of e in Π w.r.t. the channel place q is given by:*

$$depth_q^{\Pi}(e) = |\{f \in T' \mid h(f) \in {}^\bullet q^\bullet \wedge (f, e) \in F'^*\}| \,.$$

Moreover, if the process Π is clear from the context, we will write $depth_q(e)$ instead of $depth_q^{\Pi}(e)$. ◇

Proposition 1. *Let Π and $q \in Q$ be as in Definition 2. Moreover, let e and f be two distinct events of Π satisfying $\neg(e\#f)$ and $h(e), h(f) \in {}^\bullet q^\bullet$. Then $depth_q(e) \neq depth_q(f)$.*

Proof. By $\neg(e\#f)$ and Definition 1(2), we have that either eF'^+f or fF'^+e holds. Hence $depth_q(e) < depth_q(f)$ or $depth_q(e) > depth_q(f)$, respectively. □

The nets in the dashed line boxes in Figure 3(b) are two component branching processes derived from the component PT-nets of the CSPT-net in Figure 3(a). The labels are shown alongside each node, and the occurrence depth of each event connected to a (unique, in this case) channel place is shown in brackets.

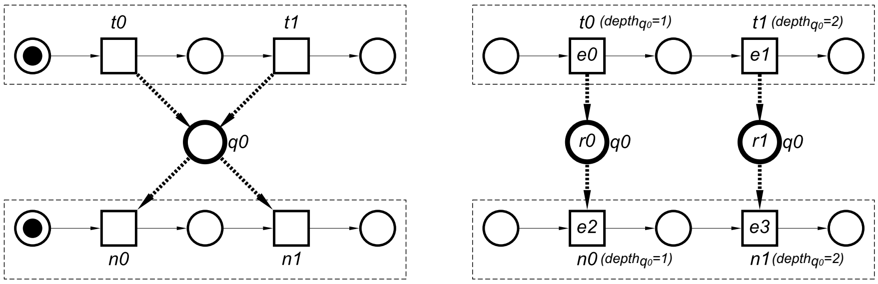

Fig. 3. (a) A CSPT-net, and (b) its branching process (event labels are shown alongside the nodes and the occurrence depths are shown in brackets).

Let us consider event e_1. Its corresponding transition t_1 is the input of channel place q_0. When searching the directed path starting at the initial state and terminating at e_1, we can find another event (viz. e_0) such that its corresponding transition is also the input of q_0. Therefore the occurrence depth of e_1, w.r.t. q_0, is $depth_{q_0}(e_1) = 2$. It intuitively represents transition t_1 passing the second token to the channel.

3.2 Non-branching processes of CSPT-nets

Similarly to the way in which CSPT-nets are extensions of PT-nets, non-branching processes of CSPT-nets are extensions of non-branching occurrence nets.

Definition 3 (non-branching process of CSPT-net). *Let $CSPT$ be as in Definition 1 with M_0 being empty.*

A non-branching process *of $CSPT$ is a tuple:*

$$CSON = (\Pi_1, \ldots, \Pi_k, Q', W', h')$$

such that each $\Pi_i = (ON_i, h_i)$ is a non-branching process of PT_i with $ON_i = (P'_i, T'_i, F'_i)$; Q' is a set of channel places; $W' \subseteq (T' \times Q') \cup (Q' \times T')$ where $T' = \bigcup T'_i$; and $h' : Q' \to Q$.

It is assumed that the following hold, where $h = h' \cup \bigcup h_i$ and $F' = \bigcup F'_i$:

1. *The ON_i's and Q' are pairwise disjoint.*
2. *For every $r \in Q'$,*
   - *$|{}^\bullet r| = 1$ and $|r^\bullet| \leq 1$; and*
   - *if $e, f \in {}^\bullet r^\bullet$, then $depth_{h(r)}(e) = depth_{h(r)}(f)$.*
3. *For every $e \in T'$, the restriction of h to ${}^\bullet e \cap Q'$ is a bijection between ${}^\bullet e \cap Q'$ and ${}^\bullet h(e) \cap Q$, and similarly for $e^\bullet \cap Q'$ and $h(e)^\bullet \cap Q$.*
4. *The relation*
$$(\sqsubset \cup \prec)^* \circ \prec \circ (\prec \cup \sqsubset)^*$$
   over T' is irreflexive, where:

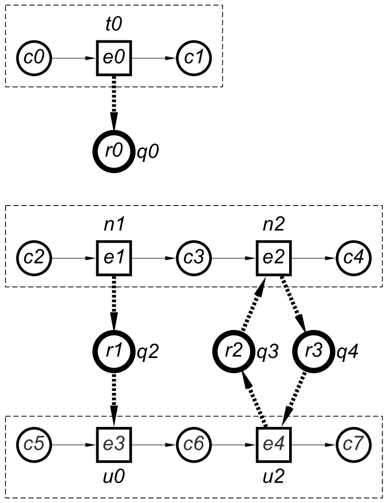

Fig. 4. A CSON-net which is a possible single run of the CSPT-net of Figure 2.

  - $e \prec f$ *if there is* $p \in \bigcup P'_i$ *with* $p \in e^\bullet \cap {}^\bullet f$*; and*
  - $e \sqsubset f$ *if there is* $r \in Q'$ *with* $r \in e^\bullet \cap {}^\bullet f$*.*

5. $h(M_{CSON}) = M_{CSPT}$*, where* M_{CSON} *is the default initial state of CSON defined as* $\bigcup M_{ON_i}$*.* ◇

The above definition extends that in [7] by allowing an infinite number of nodes, and therefore provides a general meaning of a single run of a CSPT-net. To capture the behaviour systems with complex structure, we use the the binary relation $\sqsubset$ (*weak causality*) to represent a/synchronous communication between two events (see [7]). Intuitively, the original causality relation $\prec$ represents the 'earlier than' relationship on the events, and $\sqsubset$ represents the 'not later than' relationship. In order to ensure the resulting causal dependencies remain consistent, we require the acylicity of not only each component non-branching process but also any path involving both $\sqsubset$ and $\prec$. The condition involving the depth of two events accessing the same channel place means that the tokens flowing through channel places are based on the FIFO policy, so that the size of the subsequent full representation of the behaviours of a CSPT-net is kept low.

The CSON in Figure 4 shows a non-branching processes with the labels (alongside the nodes) coming from the CSPT-net shown in Figure 2. It corresponds, e.g., to the step sequence $\lambda = \{t_0, n_1\}\{u_0\}\{n_2, u_2\}$ in the original CSPT-net.

3.3 Branching processes of CSPT-nets

We have described two classes of structured nets, i.e., CSPT-nets and CSONs. The former is a system-level class of nets providing representations of entire systems, whereas the latter is a behaviour-level class of nets representing single runs of such systems. In this section, we will introduce a new class of branching nets which can capture the complete behaviours of CSPT-nets.

Definition 4 (branching process of CSPT-net). *Let $CSPT$ be as in Definition 1 with M_0 being empty.*

A branching process *of $CSPT$ is a tuple:*

$$BCSON = (\Pi_1, \ldots, \Pi_k, Q', W', h')$$

such that each $\Pi_i = (ON_i, h_i)$ is a branching process of PT_i with $ON_i = (P'_i, T'_i, F'_i)$; Q' is a set of channel places; $W' \subseteq (T' \times Q') \cup (Q' \times T')$ where $T' = \bigcup T'_i$; and $h' : Q' \to Q$.

It is assumed that the following hold, where $h = h' \cup \bigcup h_i$ and $F' = \bigcup F'_i$:

1. *The ON_i's and Q' are pairwise disjoint.*
2. *For all $r, r' \in Q'$ with $h(r) = h(r')$, as well as for all $e \in {}^\bullet r^\bullet$ and $f \in {}^\bullet r'^\bullet$,*

$$depth_{h(r)}(e) = depth_{h(r')}(f) \iff r = r' \ .$$

3. *$BCSON$ is covered in the graph-theoretic sense by a set of non-branching processes $CSON$ of $CSPT$ satisfying $M_{CSON} = M_{BCSON}$, where the default initial state M_{BCSON} of $BCSON$ is defined as $\bigcup M_{ON_i}$.* ◇

Using arguments similar to those used in the case of the standard net unfoldings, one can show that there is a unique maximal branching process $BCSON_{CSPT}$, called the *unfolding* of $CSPT$.

A branching process of a CSPT-net consists of branching processes obtained from each component PT-net and a set of channel places. The default initial state M_{BCSON} consists of the initial states in the component branching processes. In addition, Definition 4(1) means that the component branching processes are independent, and all the interactions between them must be via the channel places. In particular, there is no direct flow of tokens between any pair of the component branching processes. Definition 4(2) implies that the occurrence depths of events inserting tokens to a channel place are the same, and are equal to the occurrence depths of events removing the tokens. Moreover, channel places at the same depth correspond to different channel places in the original CSPT-net. Finally, Definition 4(3) specifies that the label of every input and output event of a channel place in BCSON matches a corresponding transition in the original CSPT-net. In general, every node and arc in the branching process belongs to at least one non-branching process of CSPT-net (CSON). This ensures that every event in the BCSON is *executable* from the default initial state M_{BCSON} (i.e., it belongs to a step enabled in some reachable marking), and every condition and channel place is reachable (i.e., it belongs to the initial state or to the post-set of some executable events).

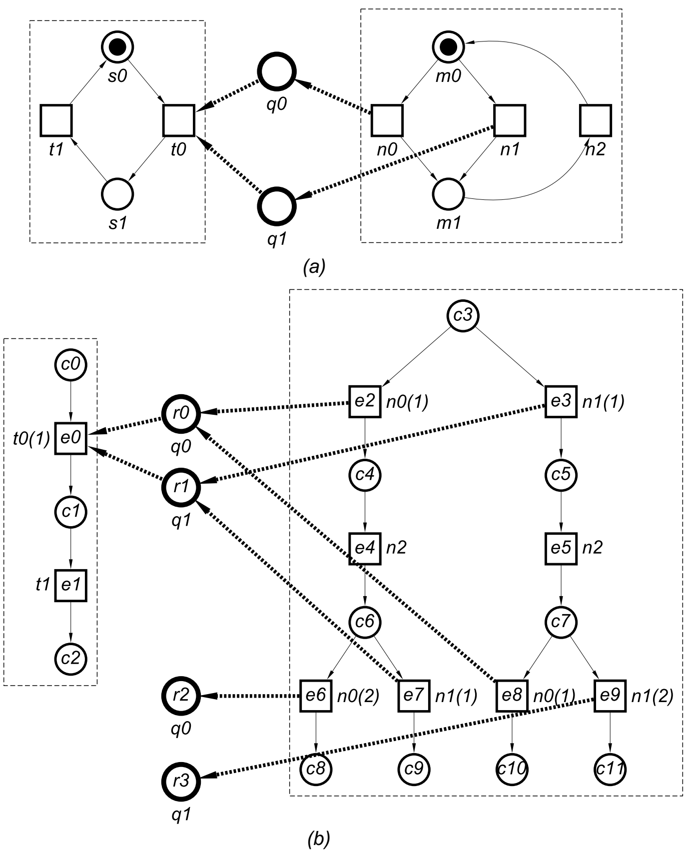

Fig. 5. (a) CSPT-net, and (b) its branching process.

Proposition 2 (safeness). *Let BCSON be as in Definition 4. Then BCSON is safe when executed from the default initial state M_{BCSON}.*
Note: This means that we treat BCSON as a CSPT*-net with the initial marking obtained by inserting a single token in each condition belonging to M_{BCSON}, and safety means that no reachable marking contains more than one token in any condition, including the channel places.*

Proof. For the conditions which are not channel places, this follows from the general properties of the branching processes of PT-nets. For the channel places, this follows from Proposition 1 and the fact that no event in a branching process of a PT-net can be executed more than once from the default initial marking. □

The nets in Figure 3(b) and Figure 5(b) are the branching processes of the CSPT-nets showing in Figure 3(a) and Figure 5(a) respectively. We can observe that every input and output event of a channel place has the same occurrence depth which represents the token flow sequence during communication between different PT-nets. For instance, in Figure 5(b) the occurrence depths of e_0, e_2 and e_8 are $depth_{q_0}(e_0) = depth_{q_0}(e_2) = depth_{q_0}(e_8) = 1$. This means of that the transitions t_0 and n_0 were involved in a first asynchronous communication.

Remark 1. A BCSON cannot, in general, be obtained by simply unfolding every component PT-net independently and appending the necessary channel places afterwards. Proceeding in such a way can lead to a net violating Definition 4(3). This is so because an executable transition in a component PT-net does not have to be executable within the context of the CSPT-net. For example, Figure 6(b) does not show a valid branching process of the CSPT-net of Figure 2. Transition n_0 in the middle PT-net of Figure 2 can never be executed since t_0 and t_1 are in conflict, and the system is acyclic. As the result, there is no n_0-labelled event in a corresponding branching process. Note that Figure 6(a) shows a valid BCSON since each event present there is executable. ◇

Remark 2. If one was only interested in marking reachability, then one might attempt to encode a CSPT-net by replacing every asynchronous channel place by a standard place and 'glue' transitions forming a synchronous event into a single one. For example, synchronised transitions n_2 and u_2 in Figure 2 could be fused into a single transition. One would then be able to apply the standard unfolding to this Petri net based representation. However, the efficiency of such an approach would suffer from the introduction of exponentially many new transitions, as well as the loss of the merging on channel places which is due to the use of occurrence depth. ◇

4 Completeness of branching processes

In this section, we introduce the concept of a complete prefix of the unfolding of a CSPT-net. The prefix is a truncated part of possibly infinite unfolding which contains full reachability information about the original CSPT-net. The idea is to consider global configurations of the unfolding taking into account single runs across different component PT-nets. Then we show that the final states of all the finite global configurations correspond to the reachable markings of original CSPT-net. Using this result, it is possible to consider a finite truncation which is sufficient to represent all reachable markings.

4.1 Global configurations

A global configuration of a BCSON consists of a set of (standard) configurations, each coming from a different component branching process, joined together by channel places.

Definition 5 (global configuration). *Let BCSON be as in Definition 4.*

A global configuration *of BCSON is a set of events*

$$C = C_1 \cup \cdots \cup C_k$$

such that each C_i is a configuration of the process Π_i, and the following hold:

1. $^\bullet C \cap Q' \subseteq C^\bullet$.
2. *The relation*
$$(\sqsubset \cup \prec)^* \circ \prec \circ (\prec \cup \sqsubset)^*$$
over C is irreflexive, where:
   - $e \prec f$ *if there is* $p \in \bigcup P_i'$ *with* $p \in e^\bullet \cap {}^\bullet f$*; and*
   - $e \sqsubset f$ *if there is* $r \in Q'$ *with* $r \in e^\bullet \cap {}^\bullet f$.

Moreover, if the configuration C is finite, then $Fin(C) = (M_{BCSON} \cup C^\bullet) \setminus {}^\bullet C$ is the final *state of C.*

The set of all global configurations of BCSON will be denoted by $Conf_{BCSON}$. ⋄

Definition 5(1) reflects the nature of a/synchronous communication between component (standard) configurations. Intuitively, if we start with an event of the global configuration which is an output event of a channel place, then there exists an input event of the same channel place that also belongs to the global configuration. Moreover, Definition 5(2) states that there are no asynchronous cycles in a global configuration.

Proposition 3 (configuration is non-branching). *Let C be a configuration as in Definition 5. Then, for all distinct $e, f \in C$, ${}^\bullet e \cap {}^\bullet f = e^\bullet \cap f^\bullet = \varnothing$.*

Proof. Suppose that ${}^\bullet e \cap {}^\bullet f \neq \varnothing$. Then by Definitions 4(1) and 5 and the definition of a configuration of a branching occurrence net, e, f belong to the same net Π_i and there is $r \in Q'$ such that $r \in {}^\bullet e \cap {}^\bullet f$. This, however, contradicts Proposition 1. As a result, ${}^\bullet e \cap {}^\bullet f = \varnothing$. The proof of $e^\bullet \cap f^\bullet = \varnothing$ is similar. □

Proposition 4 (configuration is causally closed). *Let C be a configuration as in Definition 5. Then, for every $e \in C$, $p \in \bigcup P_i'$ and $p \in e^\bullet \cap {}^\bullet f$ imply $f \in C$. Moreover, if $r \in Q' \cap {}^\bullet e$ then there is $f \in C$ such that $r \in f^\bullet$.*

Proof. Follows from the definition of a configuration of a PT-net, Proposition 1 and Definition 5. □

Since in BCSON we use the *merging* technique in the case of channel places (i.e., different events with same occurrence depth and label will link with same instance of channel place), it is possible for a channel place to have multiple inputs or outputs. Propositions 3 and 4 imply that global configuration are guaranteed to be non-branching and causally closed w.r.t. the flow relations F' and W'. Indeed, if a channel place has more than one input (or output) events, these events are in conflict w.r.t. the flow relation F'. Hence the events belong to different configurations, and each channel place in global configuration has exactly one input and no more than one output. As a result, a global configuration retains key properties of the standard configurations, and it represents a valid execution of transitions of the original CSPT-net.

Consider the branching process in Figure 5. It has a configuration $C = \{e_0, e_1, e_2, e_4, e_7\}$ which consists of two configurations $C_1 = \{e_0, e_1\}$ and $C_2 =$

$\{e_2, e_4, e_7\}$, whereas $C' = \{e_0, e_1, e_2, e_4\}$ and $C'' = \{e_0, e_1, e_2, e_4, e_6, e_7\}$ are not valid configurations (C' has non input event for the channel place r_1, while C'' includes two configurations of a single component PT-net).

Each finite configuration C has a well-defined final state determined by the outputs of the events in C. Intuitively, such a state comprises the conditions and channel places on the frontier between the events of C and events outside C. Note that a final state may contain channel places which were involved in asynchronous communications. No channel place involved in a synchronous communications can appear in $Fin(C)$, as such channel place must provide input for another event. For instance, the final state of the global configuration example above is $Fin(C) = \{c_2, c_9\}$, whereas the final state of another global configuration $C''' = \{e_2, e_4, e_6\}$ is $Fin(C''') = \{r_0, r_2, c_8\}$ which contains two asynchronous channel places.

The next result shows that a global configuration together with their outputs and the initial state form a CSON representing a non-branching process of the original CSPT-nets. And, similarly, the events of a non-branching process included in a branching one form a global configuration.

Proposition 5. *Let BCSON be as in Definition 4.*

1. *Let C be a global configuration as in Definition 5. Then $M_{BCSON} \cup C \cup C^\bullet$ are the nodes of a non-branching process of CSPT included in BCSON.*
2. *The events of any non-branching process CSON included in BCSON and satisfying $M_{CSON} = M_{BCSON}$ form a global configuration.*

Proof. (1) Let $C = C_1 \cup \dots \cup C_k$ be as in Definition 5. From the standard theory we know that, for each i, $M_{ON_i} \cup C_i \cup C_i^\bullet$ form the nodes of a non-branching process Π'_i of PT_i included in Π_i and satisfying $M_{\Pi'_i} = M_{\Pi_i}$. Define $CSON$ as composed of $\Pi'_1, \dots, \Pi'_k$, the channel places in $C^\bullet$ and the connecting arrows. We need to show that $CSON$ satisfies Definition 3.
We then observe that: Definition 3(2) follows from Proposition 3 and Definitions 5(1) and 4(2); Definition 3(3) follows from Definition 4(3); Definition 3(4) follows from Definition 5(2); and Definition 3(5) follows from Definition 5(2) and $M_{\Pi'_i} = M_{\Pi_i}$.

(2) Follows from Definition 3 and an argument reversing that carried out in part (1). □

Proposition 6. *Let C be a global configuration as in Definition 5. Then $h(Fin(C))$ is a reachable marking in the original* CSPT*-net.*

Proof. Follows from Proposition 5(1) and the properties of non-branching processes of CSPT-nets.

By combining Propositions 5 and 6, we obtain that finite global configurations provide a faithful representation of all the reachable marking of the original CSPT-net.

Theorem 1. *Let $BCSON_{CSPT}$ be the unfolding of CSPT. Then M is a reachable marking of CSPT if and only if $M = h(Fin(C))$, for some global configuration C of $BCSON_{CSPT}$.*

4.2 Complete prefixes of CSPT-nets

A complete prefix of the unfolding of a CSPT-net contains a full reachability information about the original CSPT-net. Such a property is referred to as *completeness.*

Finite complete prefixes of Petri nets were first introduced in McMillan's seminal work in order to avoid the state explosion problem in the verification of systems modelled with Petri nets. McMillan also provided an algorithm to generate a complete finite prefix of the unfolding which contains a full reachability information. Later, [3] refined McMillan's prefix construction algorithm to avoid creating prefixes larger than necessary.

The semantical meaning of completeness has been further addressed in [6], which extended it to more general properties. Basically, [6] associated completeness with some additional information, provided by the cut-off events which were only considered as an algorithm issue in the previous works. We can adapt the resulting notion to the current context as follows.

Definition 6 (completeness). *Let BCSON be as in Definition 4, and* E_{cut} *be a set of events of BCSON. Then BCSON is complete w.r.t.* E_{cut} *if the following hold:*

- *for every reachable marking* M *of* $CSPT$*, there is a finite global configuration* C *such that* $C \cap E_{cut} = \varnothing$ *and* $Fin(C) = M$*; and*
- *for each global configuration* C *of* $BCSON_{CSPT}$ *such that* $C \cap E_{cut} = \varnothing$ *and, for each event* $e \notin C$ *of* $BCSON_{CSPT}$ *such that* $C \cup \{e\}$ *is a global configuration of* $BCSON_{CSPT}$*, it is the case that* e *belongs in BCSON.*

Moreover, BCSON *is* marking complete *if it satisfies the first condition.* ◇

5 Unfolding algorithm for CSPT-net

We will now describe an algorithm for the construction of the unfolding of a CSPT-net. A key notion used by the algorithm is that of an executable event (i.e., an event which is able to fire during some execution from the default initial state) as well as that of an reachable condition or channel place (i.e., one produced by an executable event). Note that whether an event is executable in a CSPT-net is not only determined by the corresponding PT-net, but also by the behaviours of other PT-nets. This means that a component branching process in CSPT unfolding may not preserve its own unfolding structure (see Remark 1 and Figure 6(a)). In other words, there may exist events which are valid extensions in the unfolding process of a component PT-net, but become invalid when considering communication.

In particular, due to synchronous communication, it may be difficult to make sure that every extension is executable before appending it to the unfolding. Unlike the standard unfolding methods, an algorithm for CSPT-net cannot simply unfold the component branching processes adding one event at a time, and connecting it to already existing channel places. This is because a synchronous communication in CSPT unfolding forms a cycle. It is therefore impossible to add

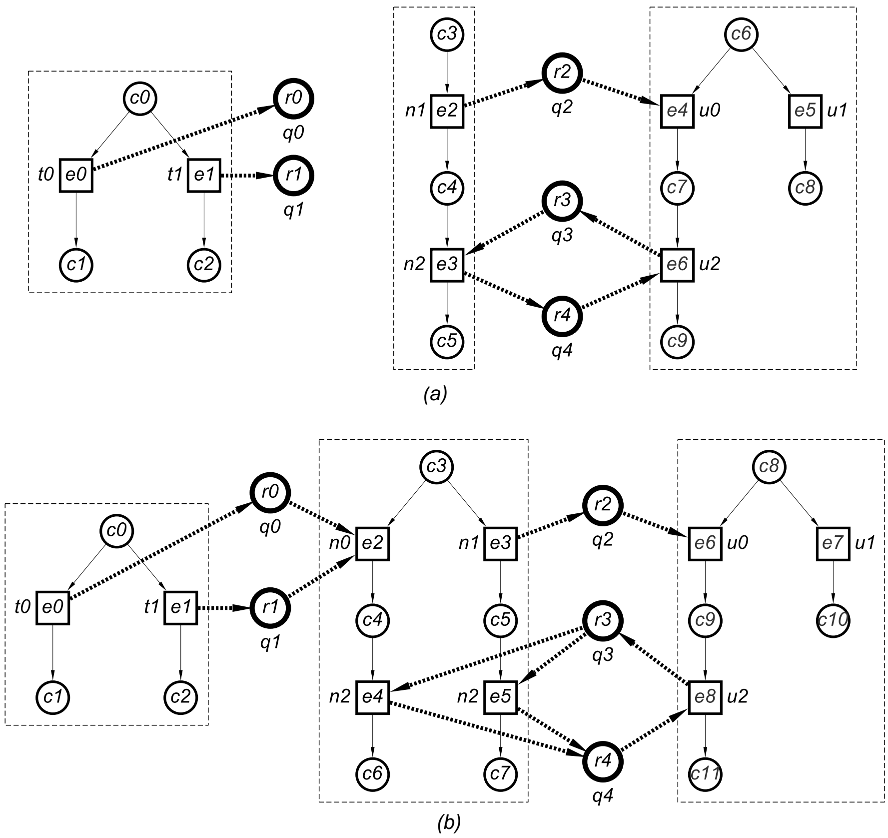

Fig. 6. (a) A valid CSPT branching process of Figure 2 (top), and (b) an invalid one (bottom).

only one of the synchronised events and guarantee its executability at the same time. Similarly, adding a synchronous event set together with all related channel places in one step may also be difficult to achieve since the use of merging may produce infinitely many events which are connected to the same channel place.

Instead, our idea is to design an algorithm which will sometimes generate non-executable events requiring tokens from channel places which have not yet been generated, in the anticipation that later on a suitable (possibly synchronous) events will provide such tokens. Roughly, the algorithm appends possible extensions together with their output conditions one by one. A new event is first marked as non-executable. The algorithm then performs an executability check for the event after constructing its a/synchronous communications. In this way, we in general obtain an 'over-approximating unfolding'. The final stage of the algorithm can then be used to remove all the non-executable nodes.

Before providing the details of the algorithm, we introduce some auxiliary notions. In what follows, we assume that $CSPT$ is as in Definition 1.

Definition 7 (local CSPT configuration). *Let $e \in C$, where C is a global configuration of $BCSON$ as in Definition 5. Then the local $CSPT$ configuration of e in C, denoted by $C[e]$, is defined as*

$$C[e] = \{f \in C \mid (f, e) \in (\prec \cup \sqsubset)^*\} ,$$

where the relations $\prec$ and $\sqsubset$ are as in Definition 5. Moreover,

$$Conf(e) = \{C[e] \mid C \in Conf_{BCSON} \wedge e \in C\}$$

is the set of all $CSPT$ local configurations of e. ⋄

The CSPT local configuration of an event e in C is the set of events that are executed before (or together with) e. In general, it consists of a configuration comprising the standard local configuration of e together with a set of standard configurations coming from other branching processes. Note that an event may have different local CSPT configurations, e.g., if one of its inputs is a channel place which has multiple input events. Each such local configuration belongs to a different non-branching process. For instance, consider a global configuration $C = \{e_0, e_1, e_2, e_4, e_7\}$ in Figure 5. The CSPT local configuration of event e_0 in C is $C[e_0] = \{e_0, e_2, e_4, e_7\}$ which involves two standard local configurations, $[e_0]$ and $[e_7]$. Moreover, we can observe that the $C[e_0]$ is not the unique local configuration of e_0, as another one is $C'[e_0] = \{e_0, e_3, e_5, e_8\}$, where $C' = \{e_0, e_1, e_3, e_5, e_8\}$.

An event may even have infinitely many local configurations. Consider again the net in Figure 5. If we continue to unfold the net, we will construct infinitely many n_0 and n_1 labelled events with occurrence depth equal to 1. All of them are input events for q_0 and q_1 labelled channel places and belong to different non-branching processes.

5.1 A/sync graphs

In order to improve the efficiency of unfolding procedure, checking for the existence of a local CSPT configuration of an event can be reduced to the problem of exploring the causal dependencies between channel places.

Below we assume that if C_i is a configuration of the unfolding of the i-th component PT-net, and $e \in C_i$ and $q \in Q$ are such that $(h(e), q) \in W$ (or $(q, h(e)) \in W$), then $r = (q, depth_q(e))$ belongs to the set of implicit channel places Q_{C_i} connected to C_i. Moreover, the label of r is q, and $(e, r) \in W_{C_i}$ (resp. $(r, e) \in W_{C_i}$) is the corresponding implicit arc.

Definition 8 (a/sync graph). *Let C_i be a configuration of the unfolding of the i-th component* PT*-net.*

Then the a/sync graph *of C_i is defined as:*

$$\mathcal{G}(C_i) = (Q_{C_i}, \widehat{\prec}_{C_i}, \widehat{\sqsubset}_{C_i})$$

where $\widehat{\prec}_{C_i}, \widehat{\sqsubset}_{C_i}$ are two binary relations over Q_{C_i} such that, for every $r, r' \in Q_{C_i}$:

– *$r \mathrel{\widehat{\prec}}_{C_i} r'$ if there are two distinct $e, f \in C_i$ such that $(r, e), (f, r') \in W_{C_i}$, and e precedes f within C; and*
– *$r \mathrel{\widehat{\sqsubset}}_{C_i} r'$ if there is $e \in C_i$ with $(r, e), (e, r') \in W_{C_i}$.* $\diamond$

$\mathcal{G}(C_i)$ captures relationships between the input and output channel places of a configuration of the unfolding of an individual component system. Its nodes are the channel places involved in C_i. Moreover, $r \mathrel{\widehat{\prec}}_{C_i} r'$ if there is a path from r to r' involving more than one event of C_i, and $r \mathrel{\widehat{\sqsubset}}_{C_i} r'$ if r is an input and r' an output of some event in C_i.

Figure 7(a) shows the unfolding of each component PT-net of Figure 2 together with their input and output channel places. By exploring the relations between those channel places, we are able to generate a/sync graph for any configuration. For example, Figure 7(b) shows five a/sync graphs of the configurations derived from Figure 7(a), where the relations $\widehat{\prec}_{C_i}$ and $\widehat{\sqsubset}_{C_i}$ are represented by solid arcs and thick dashed arcs, respectively. For the left-hand side PT-net Π_1, we have that:

$$\mathcal{G}(C_1) = (\{r_0\}, \varnothing, \varnothing) \qquad \mathcal{G}(C'_1) = (\{r_1\}, \varnothing, \varnothing)$$

The a/sync graphs of the configurations in Π_2 are:

$$\mathcal{G}(C_2) = (\{r_2, r_3, r_4, r_5\}, \{(r_2, r_4), (r_3, r_4)\}, \{(r_5, r_4)\})$$
$$\mathcal{G}(C'_2) = (\{r_6, r_7, r_8\}, \varnothing, \{(r_8, r_7)\}$$

and for the right-hand side PT-net Π_3, we have that:

$$\mathcal{G}(C_3) = (\{r_9, r_{10}, r_{11}\}, \{(r_9, r_{11})\}, \{(r_{10}, r_{11})\})$$

Given a set of a/sync graphs $\mathcal{G}(C_1), \ldots, \mathcal{G}(C_k)$ extracted for the k component systems, we call these graphs compatible if all inputs are produced and there is no cycle involving $\widehat{\prec}$.

Definition 9 (compatibility of a/sync graphs). *Let C_i $(i = 1, \ldots, k)$ be a configuration of the unfolding of the i-th component* PT*-net, and $\mathcal{G}(C_i) = (Q_{C_i}, \widehat{\prec}_{C_i}, \widehat{\sqsubset}_{C_i})$.*

Then $C_1, \ldots, C_k$ are compatible *configurations if the following hold:*

1. *if $(r, e) \in W_{C_i}$ then that there is $j \neq i$ such that $r \in Q_{C_j}$; and*
2. *the relation*
$$(\widehat{\sqsubset} \cup \widehat{\prec})^* \circ \widehat{\prec} \circ (\widehat{\prec} \cup \widehat{\sqsubset})^*$$
is irreflexive, where $\widehat{\prec} = \bigcup \widehat{\prec}_{C_i}$ and $\widehat{\sqsubset} = \bigcup \widehat{\sqsubset}_{C_i}$. $\diamond$

In Figure 7, configurations C_1, C'_2, C_3 are compatible since the q_3-labelled input channel place r_8 in $\mathcal{G}(C'_2)$ is present in $\mathcal{G}(C_3)$ (i.e., r_{11}), and the input channel places r_9, r_{10} (labelled by q_2 and q_4 respectively) in $\mathcal{G}(C_3)$ are all present in $\mathcal{G}(C'_2)$. On the other hand, we can observe that there are no compatible configurations which involve C_2, i.e., neither configurations C_1, C_2, C_3 nor C'_1, C_2, C_3 are compatible. This is because the producers of r_2 and r_3 are in conflict in Π_1.

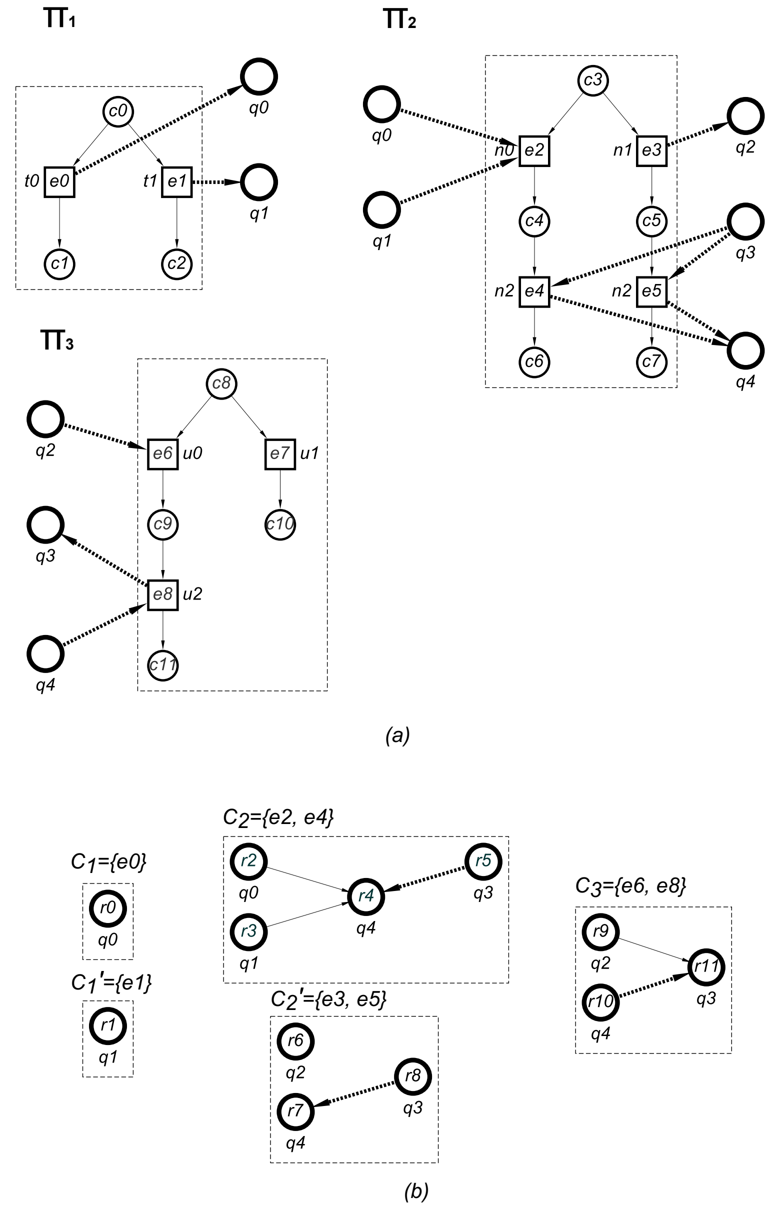

Fig. 7. (a) unfoldings of three component PT-nets of Figure 2 (together with their implicit channel places), and (b) a/sync graphs of configurations derived from these unfoldings.

Theorem 2. *Let $C_1, \ldots, C_k$ be configurations of the unfoldings of the component* PT*-nets, and $C = C_1 \cup \cdots \cup C_k$. Then C is a global configuration if and only if $C_1, \ldots, C_k$ are compatible.*

Proof. ($\Longrightarrow$) If C is a global configuration then, Proposition 4, every input channel place of a global configuration C is produced in C, and from Definition 1(2), the producer e and consumer f belong to separate configurations. Hence Definition 9(1) holds. Moreover, Definition 9(2) follows from Definition 5(2).

($\Longleftarrow$) We observe that Definition 5(1) and Definition 5(2) respectively follow from Definition 9(1) and Definition 9(2). □

Therefore, one can obtain the CSPT local configurations of an event e by checking whether there are compatible configurations $C_1, \ldots, C_k$ such that e belongs to one of them. Such a task can be made efficient by working with the graphs $\mathcal{G}(C_1), \ldots, \mathcal{G}(C_k)$. In fact, one can just check those configurations which have dependencies on e.

5.2 Unfolding algorithm

The unfolding algorithm we are going to present significantly differs from the existing net unfolding algorithms. The key difference is that during the unfolding procedure we will be constructing nodes and connections which will not necessarily be the part of the final unfolding. This is due to the presence of synchronous communication within our model. More precisely, in the net being constructed there will be *executable* and *non-executable* events and conditions. The former will definitely be included in the resulting unfolding, whereas the latter cannot be yet included due to the absence of event(s) which are needed for communication. If, at some later stage, the missing events are generated, then the previously non-executable event and the conditions (and channel places) it produced become executable.

Although the net *Unf* generated by the algorithm may not strictly speaking be a branching process during its creation, we will as far as it is possible treat it as such. In particular, we will call an event e *executable* if $Conf(e) \neq \varnothing$. This happens if we have generated enough events to find at least one local CSPT configuration of e in *Unf*.

Intuitively, an executable event is the event belonging to at least one single run of a BCSON. For the example net in Figure 6(b), e_6 is an executable event since there exists a local CSPT configuration of e_6: $C[e_6] = \{e_0, e_3, e_6\}$, where $C = \{e_0, e_3, e_6\}$. On the other hand, event e_2 is non-executable because it does not have any local configuration (we have seen the example of Figure 7 that there are no compatible configurations which involve e_2). Therefore, Figure 7(b) is not a valid CSPT branching process since according to Definition 4(3) every event in *BCSON* is executable. If we remove e_2 together with its successors, then all events in the new net become executable indicating the net is a valid BCSON (Figure 6 (a)).

Proposition 7. *Let e be an executable event in BCSON. Then each event appearing in $Conf(e)$ is executable.*

Proof. From Definition 7 it follows that $f \in Conf(e)$ implies that there exist a global configuration C such that $e, f \in C$. Hence $Conf(f) \neq \varnothing$, and so the result follows from the definition of executable events. □

Algorithm 1 (unfolding of CSPT-net)

```
input: CSPT — CSPT-net
output: Unf — unfolding of BCSON

nonexe ← ∅
Unf ← the empty branching process
add instances of the places in the initial marking of CSPT to Unf
add all possible extensions of Unf to pe

while pe ≠ ∅ do
  remove e from pe
  addConnections(e)
  if Conf(e) ≠ ∅ then
    for all event f in configurations of Conf(e) do
      remove f and all its output conditions from nonexe (if present there)
    add all possible extensions of Unf to pe

delete the nodes in nonexe together with adjacent arcs from Unf
```

The procedure for constructing the unfolding of a CSPT-net is presented as Algorithm 1.

The first part of the algorithm adds conditions representing the initial marking of the CSPT-net being unfolded. Notice that the set *nonexe* of non-executable events and conditions is set to empty. It also adds possible extensions to the working set *pe*. The concept of a non-executable condition greatly improves the efficiency of the above algorithm since a *possible extension* of *Unf* is a pair $e = (t, B)$ with $h(e) = t$ where t is a transition of $CSPT$, and B is a set of conditions of *Unf* such that:

- B is a co-set in one of the subnets of *Unf* and $B \cap nonexe = \varnothing$;
- $h(B)$ are all the input non-channel places of t; and
- $(t, B) \notin pe$ and *Unf* contains no t-labelled event with the non-channel place inputs B.

The pair (t, B) is an event used to extend BCSON without considering channel places. We use the standard condition of a possible extension to choose events that can be added to a component branching process (i.e., $h(B) = {}^\bullet t \cap P'$), while constructing the related a/synchronous communications in a separate step.

In such a way, the complexity of appending groups of synchronous events is significantly reduced. Note that a possible extension e has precisely determined channel place connections since the depth values are fully determined.

Algorithm 2 (adding new event and a/sync connections)

---

procedure addConnections (**input:** $e = (t, B)$)
  add e to *Unf* and *nonexe*
  create and add all the standard post-conditions of e to *Unf* and *nonexe*
  for all channel place $q \in {}^{\bullet}t^{\bullet}$ **do**
    let $r = (q, k)$ where $k = depth_q(e)$
    if there is no $r = (q, k)$ in *Unf* **then**
      add q-labelled channel place r to *Unf* and *nonexe*
    add a corresponding arc between r and e

Algorithm 2 provides the details of appending a possible extension e to BCSON as well as constructing related channel place structure after removing e from *pe*. Each new extension and its output conditions are immediately marked as non-executable. The conditions in *nonexe* set also indicate that they are unable to be used for deciding any further possible extension. In this way we can avoid any unnecessary extension and make sure the predecessors of every new event is executable.

The procedure then creates the a/synchronous communications of the input event if it is required. Given an event e, for every input or output channel place q of its corresponding transition $h(e)$ in the original CSPT-net, we search in *Unf* for the matching channel place (i.e., its label is q and its depth value equals to the occurrence depth of e). Then we create a direct connection if such a channel place exists. Otherwise, we add a new instance of the channel place together with the corresponding arc.

After adding the implicit channel places connected to e (or creating the connection for those which already existed) together with the corresponding arcs, we are able to obtain the local configuration of e by looking for compatible configurations $C_1, \ldots, C_k$ of the component nets (which may contain non-executable events) such that e belongs to one of the C_i's. If e is executable ($Conf(e) \neq \varnothing$), we make all non-executable events in $Conf(e)$ together with their post-conditions executable (see Proposition 7). We also generate new potential extensions (each such extension must use at least one of conditions which have just been made executable). Then another waiting potential extension (if any) is processed.

The algorithm generally does not terminate when the original CSPT-net is not acyclic, and the non-executable nodes are removed at the end of the algorithm.

We now use the CSPT-net in Figure 2 as an example to illustrate the algorithm. The unfolding process is shown in Figure 8. It starts by appending instances of the initial marking s_0, m_0 and v_0 of the CSPT-net to *Unf*. Then we go to the first iteration which adds a possible extension $e_0 = (t_0, \{c_0\})$ together with its post-condition to both *Unf* and *nonexe* (nodes belonging to *nonexe* are shaded). It is not possible to confirm the executability of e_0 at this point due to the absence of a suitable channel place: to generate related communications, we search in the *Unf* for a q_0-labelled and $depth_{q_0}(e_0) = 1$ channel place. Apparently, there is no such place, and so we need to add a copy of q_0 to the unfolding and connect it with e_0. After executing *addConnection*(e_0) procedure, we find that there is a local configuration of e_0 (which is in fact $\{e_0\}$ itself) by building a/sync graph and verifying compatibility. This local configuration indicates $h(e_0)$ is executable in one of the single runs of the original CSPT-net, and hence is also an executable event in the corresponding unfolding. Therefore, we are allowed to remove e_0 and its output condition c_1 from *nonexe*. c_1 then can be used to generate further extensions after our first complete iteration.

Stage C shows the result after three iterations (i.e., appending e_0, e_1 and e_2). Notice that in the *AddConnection*(e_2) procedure, it is possible to add only arcs from existing channel places r_0 and r_1 to the chosen event e_2. This is so because suitable channel places were already added in the previous iterations. Moreover, e_2 and c_4 remain in *nonexe* after their own iteration since there is no local configuration containing e_2. Such a condition cannot be used for generating new possible extension until it becomes executable.

Stage D and E illustrate the way to generate synchronous communication between e_4 and e_7. After adding e_4 to the unfolding, we first create 'half' of the synchronous communication, i.e., we add channel places r_3, r_4 and the relations (r_3, e_4), (e_4, r_4). It can be observed that there is no compatible configurations involving e_4 at the moment since the producer of r_3 is missing. Therefore, e_4 is a non-executable event. However, this non-executable event and its post-condition can be removed from *nonexe* after adding another 'half' of the synchronous communication i.e., event e_7 and the related connections. More precisely, after *AddConnection*(e_7), we have $nonexe = \{e_2, c_4, e_7, c_{10}\}$. We can then find compatible configurations $C_1 = \{e_1\}, C_2 = \{e_3, e_4\}, C_3 = \{e_5, e_7\}$ which contain both non-executable events e_4 and e_7. These two events and their output conditions can then be removed from *nonexe*.

After adding all possible extensions from *pe* to *Unf* (in this example, the unfolding is finite states since the original CSPT-net is acyclic), the final stage is to remove from the unfolding all the nodes in *nonexe*. As a result, we obtain a correct unfolding by removing e_2 and c_4 (it is the net shown in Figure 6(a)).

6 Conclusions and Future Work

We introduced the concept of a branching process of a CSPT-net — a new class of branching nets which can capture complete behaviours of CSPT-nets. The model extends the standard branching processes by combining multiple component branching processes with channel places. We then formulated the property of completeness for BCSONs. The concept relies on the notion of a global configuration which describes a single run of transitions crossing different PT-nets.

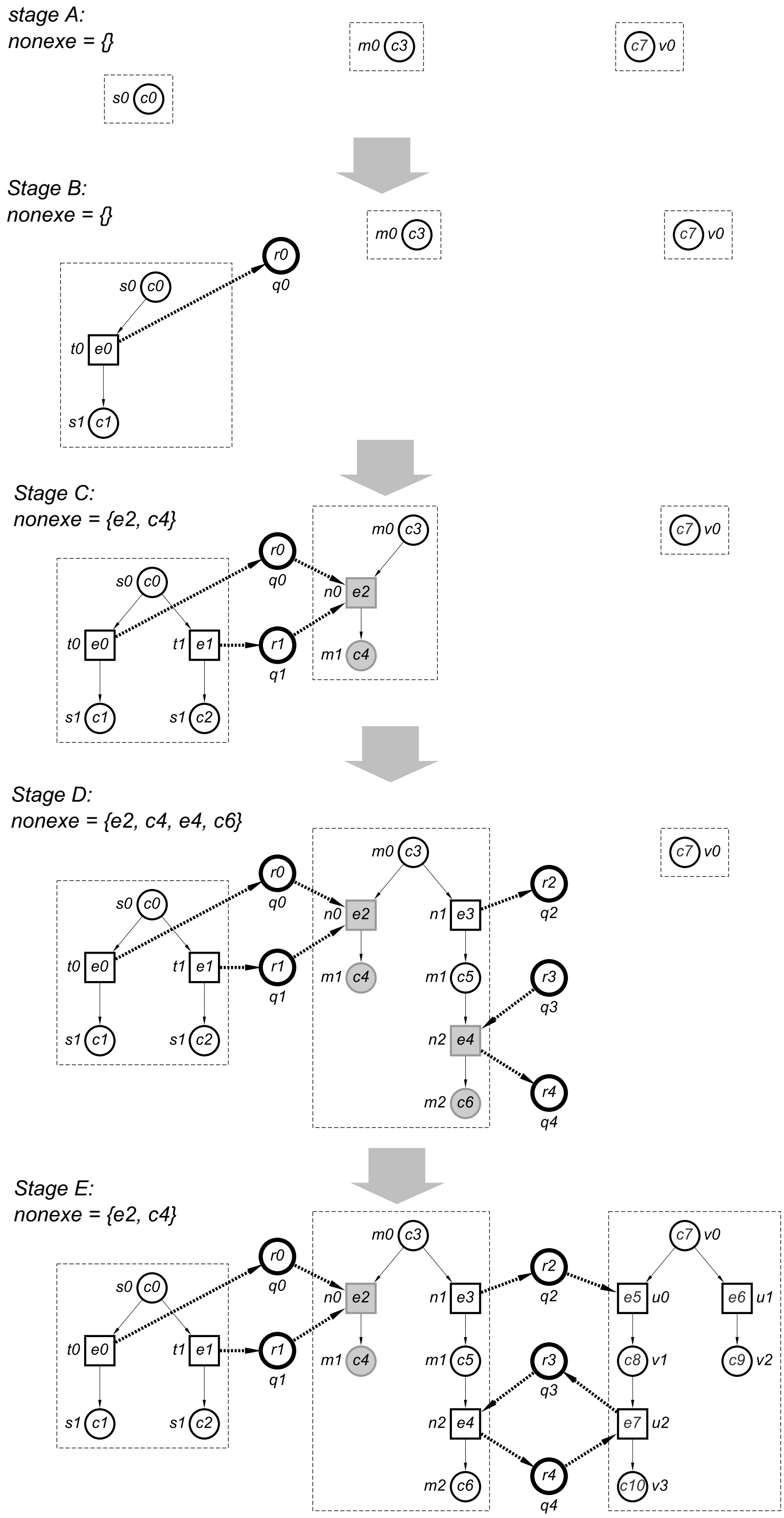

Fig. 8. Unfolding the CSPT-net of Figure 2: (Stage A) starting point of the unfolding, (Stage B) after first complete iteration, (Stage C) chosen event e_2 is non-executable, (Stage D) chosen event e_4 is non-executable, (Stage E) e_4 becomes executable due to the missing event is generated.

Our investigation has led to an algorithm for constructing the unfolding of a CSPT-net which is its unique maximal BCSON. A central part of the unfolding algorithm is the test of whether a non-executable event can become executable. This is done by checking whether there are global configurations such that the event belongs to one of them. Moreover, only conditions marked as executable can to be used for constructing new possible extensions, so that we always generate further extensions on the basis of executable elements.

The algorithm presented in this paper is based on standard unfolding method, which essentially works by appending possible extension one by one. A potentially very efficient approach for the construction of the unfolding could be to use the parallel unfolding technique [4]. One can, for example, unfold each component branching process in parallel, by temporarily ignoring any a/synchronous issues. The procedures of appending channel places as well as executability checking (removing unnecessary events) would proceed in a separate step. In this way, we might significantly improve the efficiency of the algorithm since different component net unfoldings can be constructed on multiple computer processors.

In future we intend to explore the generation of finite complete prefixes of CSPT-nets. In the case of PT-nets, this relies on the notion of *cut-off* events, which are roughly events in the unfolding that produce a marking already produced by other events with smaller histories. In CSPT unfolding, we may instead use global configurations to determine the repetition of markings.

In general, it is impossible to generate a finite complete prefix of the unfolding of a CSPT-net even if the component PT-nets are safe. The reason is that the channel places linking the component PT-nets can be unbounded due to asynchronous communication. However, if all communications are synchronous, this is no longer a problem.

We intend to implement the CSPT model and its analysis tools in Workcraft platform [12]. The platform provides a flexible common underpinning for graph based models. The facilities for entering, editing, validating, visualising and simulating structured occurrence nets have been implemented in Workcraft [10].

Acknowledgement

We would like to thank Victor Khomenko for several very helpful comments on the previous versions of this paper.

References

1. Best, E., Fernández, C.: Nonsequential Processes: A Petri Net View, vol. 13 of EATCS Monographs in Theoretical Computer Science. Springer-Verlag (1988)
2. Engelfriet, J.: Branching processes of Petri nets. Acta Informatica 28(6), 575–591 (1991)
3. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan's unfolding algorithm. In: Formal Methods in System Design. pp. 87–106. Springer-Verlag (1996)

4. Heljanko, K., Khomenko, V., Koutny, M.: Parallelisation of the Petri net unfolding algorithm. In: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 371–385. TACAS '02, Springer-Verlag, London, UK, UK (2002)
5. Khomenko, V., Kondratyev, A., Koutny, M., Vogler, W.: Merged processes: A new condensed representation of Petri net behaviour. Acta Informatica 43(5), 307–330 (2006)
6. Khomenko, V., Koutny, M., Vogler, W.: Canonical prefixes of Petri net unfoldings. Acta Inf. 40(2), 95–118 (2003)
7. Kleijn, J., Koutny, M.: Causality in structured occurrence nets. In: Dependable and Historic Computing. vol. 6875, pp. 283–297. Springer Berlin Heidelberg (2011)
8. Koutny, M., Randell, B.: Structured occurrence nets: A formalism for aiding system failure prevention and analysis techniques. Fundamenta Informaticae 97(1), 41–91 (Jan 2009)
9. Li, B.: Branching processes of communication structured PT-nets. In: Proceeding. vol. 13th International Conference On Application of ConCurrency to System Design (ACSD), pp. 243–246 (2013)
10. Li, B., Randell, B.: Soncraft user manual. Tech. Rep. CS-TR-1448, School of Computing Science, Newcastle University (Feb 2015)
11. McMillan, K.L., Probst, D.: A technique of state space search based on unfolding. Formal Methods in System Design 6(1), 45–65 (Jan 1995)
12. Poliakov, I., Khomenko, V., Yakovlev, A.: Workcraft – A framework for interpreted graph models. In: Applications and Theory of Petri Nets, pp. 333–342. Springer Berlin Heidelberg (Jun 2009)
13. Randell, B.: Occurrence nets then and now: the path to structured occurrence nets. In: Applications and Theory of Petri Nets. pp. 1–16. Springer Berlin Heidelberg (Jun 2011)
14. Randell, B., Koutny, M.: Failure: their definition, modelling and analysis. In: Theoretical Aspects of Computing–ICTAC 2007. pp. 260–274. Springer (Sep 2007)
15. Watson, P.: A multi-level security model for partitioning workflows over federated clouds. In: IEEE 3rd International Conference on Cloud Computing Technology and Science, CloudCom 2011, Athens, Greece, November 29 - December 1, 2011. pp. 180–188 (2011)