



BIROn - Birkbeck Institutional Research Online

Sroka, J. and Hidders, Jan (2017) On determining the AND-OR hierarchy in workflow nets. *Fundamenta Informaticae* 156 (1), pp. 95-123. ISSN 0169-2968.

Downloaded from: <https://eprints.bbk.ac.uk/id/eprint/30940/>

Usage Guidelines:

Please refer to usage guidelines at <https://eprints.bbk.ac.uk/policies.html> or alternatively contact lib-eprints@bbk.ac.uk.

On Determining the AND-OR Hierarchy in Workflow Nets

Jacek Sroka

Institute of Informatics

University of Warsaw, Poland

sroka@mimuw.edu.pl

Jan Hidders

Vrije Universiteit Brussel, Belgium

jan.hidders@vub.be

Abstract. This paper presents a notion of reduction where a WF net is transformed into a smaller net by iteratively contracting certain well-formed subnets into single nodes until no more of such contractions are possible. This reduction can reveal the hierarchical structure of a WF net, and since it preserves certain semantic properties such as soundness, can help with analysing and understanding why a WF net is sound or not. The reduction can also be used to verify if a WF net is an AND-OR net. This class of WF nets was introduced in earlier work, and arguably describes nets that follow good hierarchical design principles.

It is shown that the reduction is confluent up to isomorphism, which means that despite the inherent non-determinism that comes from the choice of subnets that are contracted, the final result of the reduction is always the same up to the choice of the identity of the nodes. Based on this result, a polynomial-time algorithm is presented that computes this unique result of the reduction. Finally, it is shown how this algorithm can be used to verify if a WF net is an AND-OR net.

1. Introduction

Petri nets [18] are one of the most popular and well studied formalisms for modelling processes. Their graphical notation is easy to understand, but at the same time concrete and formal, which allows for reasoning over the complex systems that are being modelled. Petri nets are especially useful for business processes and business workflows for which a specific class of Petri nets, called *workflow nets*, was introduced [27, 28]. Even though other notations are used in most industrial process modelling tools like Business Process Modeling Notation (BPMN) [17], Business Process Execution Language (BPEL)

or Event-driven Process Chain (EPC) [13], the control flow aspect of the models expressed in those notations can be translated to workflow nets. At the same time workflow nets are considered to be the goto formalism for workflow analysis, like detecting possible problems, e.g., existence of deadlocks or livelocks, and for investigating the principles of workflow modelling without focusing on a particular language.

Workflow models that lack those problems are called *sound*, and the first definition of workflow-net soundness was proposed by van der Aalst in [27]. Quickly several alternative definitions of soundness emerged, varying in strictness and verification difficulty. Examples of these are weak soundness [14, 15], relaxed soundness [6], lazy soundness [21], k -soundness and generalised soundness [32, 31], up-to- k -soundness [30] and substitution soundness [23]. Informally, the original notion of soundness guarantees two properties of the net. First, that if we initiate the workflow net correctly, then no matter how the execution proceeds, we can always end up in a proper final state. Second, that every subtask can be potentially executed in some run of the workflow. An overview of the research on the different types of soundness of workflow nets and their decidability can be found in [29].

In earlier research [23] we have proposed a new notion of soundness, namely the substitution soundness or sub-soundness for short. It is similar to k - and $*$ -soundness studied in [32], but captures exactly the conditions necessary for building complex workflow nets by following a structured approach where subsystems with multiple inputs and outputs are used as building blocks of larger systems. As was shown in that research, it is not enough for such subsystems to be classically sound by themselves. It may be the case, for example, that if a sound WF net is used inside another sound WF net, then the nested WF net is used to execute several simultaneous computations which can interfere and cause the whole WF net to become unsound. Or it can be that partial results, represented by tokens in the output places of the nested WF net, are consumed prematurely by the containing WF net before the nested WF net has finished properly which disrupts it. This is prevented by the notion of substitution soundness (or sub-soundness), which is informally defined as follows: a WF net is sub-sound iff after initiating it with k tokens in every input place and letting it execute it will always be able to finish by producing k tokens for every output place even if during the run the output tokens are removed by some external transitions.

Although stronger soundness properties may be desirable, they are often also more difficult to verify. For this reason, a method is introduced in [23] for systematically constructing workflow nets so that they are guaranteed to satisfy the sub-soundness property. This method is in principle, and in effect, similar to methods employed in software engineering, where complexity is tackled by separation of concerns and encapsulation, and systems are divided into building blocks such as modules, objects and functions, which in turn can be decomposed further.

We follow those good practices in the context of workflow nets where they, like in software engineering, allow to avoid common pitfalls. Similarly to general programming languages, also for workflow nets, patterns and anti-patterns have been published [28, 25]. Also similarly to general programming, it is beneficial to organise the workflow models in a structured way. In programming the ideas of using macros, subroutines, procedures, functions, and later on, classes, proved that even extremely complex systems can be programmed and maintained in a practical and effective manner. Such structuration was successfully applied to designing complex Petri nets [37, 24, 7, 1, 8, 19] and workflow nets [2, 26, 23]. As with general programming, the system is composed of small, separated fragments, which are easier to understand and maintain. Fragments can include invocations of other fragments, which can include other nested fragments, and so on.

The class of nets we introduced in [23] is called the class of AND-OR nets (see Section 3). This class

is larger and more general than other classes of workflow nets generated with a similar type of structural approach, as presented for example in [32, 2]. Apart from studying conditions necessary for structured workflow systems to be $*$ -sound, it was shown in [23] that all AND-OR nets indeed are sub-sound. In this paper we continue this line of research and introduce a method to determine the hierarchical structure of a WF net, or parts of it, that was not necessarily designed in such a structured way. In [23] the AND-OR nets were defined as all the nets that can be constructed with a top-down refinement procedure, by using nets of certain basic classes similar to S/T systems. In this paper we show that at the same time AND-OR nets that were not necessarily constructed in such a way, can be analysed to determine a refinement hierarchy with a bottom-up reduction procedure that contracts subnets of the basic AND-OR classes. Moreover, it is shown that finding occurrences of such subnets can be done in polynomial time.

A key result in this paper is that the procedure of contracting subnets of the basic AND-OR classes is confluent and therefore the reduction will always return the same result, independent of how the subnets were selected for contraction. This can be used to turn the procedure into a polynomial algorithm and therefore a tractable method for determining an AND-OR refinement tree. Next to that, it can also be determined if a net is an AND-OR net by checking if the reduction procedure reduces it to a one-node WF net. If a net is positively identified as an AND-OR net, it is consequently also guaranteed to be sub-sound and consequently $*$ -sound, i.e., can be used as a building block of larger systems. An example application of this result would be a scenario where a process modeller constructs a complex model from sub-models published in some repository. He or she may want to make sure that the sub-models follow good design principles and are sub-sound, which means that they can be safely used as building blocks of a composite model. The repository can contain models for subunits in some organisational structure, e.g., models for faculties of an university or departments of a company or even models from some global repository of socially shared workflows, which appear in e-science [5].

The reduction algorithm not only can be used for AND-OR nets, but also for the analysis of the soundness and the structure of other workflow nets. It can help the user with finding problems causing unsoundness. More concretely, if the result of AND-OR verification is negative, then the reduction algorithm stops without reaching a one-node WF net. This resulting net can serve as a condensed version of the original net and point the user to the source of the problem in the design. Note that a WF net may be not an AND-OR net, but still be $*$ -sound or sub-sound. We conjecture, but have not proven, that to verify $*$ -soundness or sub-soundness of an arbitrary net, it is enough to verify $*$ -soundness or sub-soundness of the net resulting from AND-OR verification procedure. The contractions used in our algorithm would have to be proven to preserve $*$ -soundness or sub-soundness, similarly as for example rules of [16] preserve liveness and boundedness. This would give a symmetric and probably similarly laborious result to [23], where it was shown that substitutions of AND-OR nets into AND-OR nets preserve sub-soundness, from which it follows that they also preserve $*$ -soundness. The reduced net, resulting from AND-OR net verification procedure, could then undergo a proper soundness verification with similar methods as in [35, 36, 20, 9]. Furthermore, limiting the size of the verified net with hierarchical methods can be helpful for users struggling with understanding the reasons for unsoundness of workflow nets. That this is often a problem, even when using automated verification tools, as for example reported in [10].

Finally, the result of the reduction procedure can be helpful in better understanding the workflow. Similar benefits are discussed in [11] for control flow graphs which are well-known representation of the sequential control flow structure of programs and have multitude of applications. Graph reductions make it easier for software engineers to analyse the control flow in large graphs representing functions, sets of function or even complete programs. Furthermore, as a byproduct of a successful reduction of

a workflow net, a tree structure describing the nesting of the fragments of the net can be determined. As with similar methods [2, 4, 3], which deal with workflow net class which is a proper subclass of AND-OR nets, such a tree structure can be used for modelling recovery regions or determining sound markings, or just for better understanding the structure of the workflow net and its properties. The latter can for example help with determining how parts of the workflow can best be distributed to independent organisational units or to different servers in case of workflows representing computations, e.g., as in scientific workflows.

There has already been similar work where a set of heuristics is proposed to find appropriate decomposition boundaries, which results in a refinement tree for a given graph. In [34] it is however purely syntax-based, where our method is linked to particular notions of soundness. Moreover, it concentrates on workflow nets with only one input node and one output node, where we allow there to be more. There have also several results where similar methods were used to do or speed up classical soundness validation. In [36] structural reductions were used as pre-processing which reduces the net size and then S-coverability analysis as well as a form of state space exploration based on coverability trees for soundness validation. In [20] the nets are decomposed based on connectivity property of graphs. Then, the discovered compositional structure of a model is used for reasoning on soundness of the whole system. The paper also reports on findings from applying such approach to validate soundness of an industry model collection. Finally, [9] reports on investigation of three approaches implemented in three different tools based on 735 industrial business process models. The goal of the case study was to validate soundness and report the problems with workflow design to the user. All the discussed methods used some form of decomposition to deal with the complexity. As with [33] also [36, 20, 9] concentrate on workflow nets with only one input node and one output node and the results do not easily generalize to nets not limited in this way. Also the classical notion of soundness is used in all the cases which is more limited than the *-soundness and sub-soundness we use here.

The outline of the remainder of this paper is as follows. In Section 2 the basic terminology of WF nets and their semantics is introduced. In Section 3 the class of AND-OR nets is introduced, based on the notions of place and transition substitution, where a node is replaced with a WF net. In Section 4 the notion of the reduction of a WF net is introduced, which is a procedure where repeatedly certain well-formed subnets of WF nets are contracted into single nodes. It is discussed how this reduction process is confluent in the sense that it returns a unique result up to the choice of the identity of the nodes. This is based on the observation that the process is locally confluent, and since the proof of this observation is quite involved, it is presented separately in Section 5. In Section 6 a concrete polynomial algorithm for computing the result of the reduction is presented, and it is shown how it can be used to verify if a WF net is an AND-OR net. Finally, in Section 7 a summary of the results is given, and potential future research directions are discussed.

2. Basic terminology and definitions

Let S be a set. A bag (multiset) m over S is a function $m : S \rightarrow \mathbb{N}$. We use $+$ and $-$ for the sum and the difference of two bags and $=, <, >, \leq, \geq$ for comparisons of bags, which are defined in the standard way. We overload the set notation, writing \emptyset for the empty bag and \in for the element inclusion. We list elements of bags between brackets, e.g. $m = [p^2, q]$ for a bag m with $m(p) = 2$, $m(q) = 1$, and $m(x) = 0$ for all $x \notin \{p, q\}$. The shorthand notation $k.m$ is used to denote the sum bag m of k times.

The size of a bag m over S is defined as $|m| = \sum_{s \in S} m(s)$.

A *Petri net* is defined as usual as a tuple $N = (P, T, F)$ where P is a finite set of places, T is a finite set of transitions such that $P \cap T = \emptyset$ and $F \subseteq (T \times P) \cup (P \times T)$ the set of flow edges. We will refer to the elements of $P \cup T$ also as *nodes* in Petri net. We say that the *type* of a node is *place* or *transition* if it is in P or T , respectively.

A path in a net is a non-empty sequence (n_1, \dots, n_m) of nodes where for all i such that $1 \leq i \leq m-1$ it holds that $(n_i, n_{i+1}) \in F$. Markings are states (configurations) of a net and the set of markings of $N = (P, T, F)$ is the set of all bags over P and is denoted as \mathbf{M}_N . Given a transition $t \in T$, the preset $\bullet t$ and the postset $t\bullet$ of t are the sets $\{p \mid (p, t) \in F\}$ and $\{p \mid (t, p) \in F\}$, respectively. In a similar fashion we write $\bullet p$ and $p\bullet$ for pre- and postsets of places, respectively. To emphasise the fact that the preset (postset) is considered within some net N , we write $\bullet_N a$, $a\bullet_N$. We overload this notation by letting $\bullet a$ ($a\bullet$) also denote the bags of nodes that (1) contain all nodes in the preset (postset) of a exactly once and (2) contains no other nodes. A transition $t \in T$ is said to be enabled at marking m iff $\bullet t \leq m$. For a net $N = (P, T, F)$ with markings m_1 and m_2 and a transition $t \in T$ we write $m_1 \xrightarrow{t}_N m_2$, if t is enabled at m_1 and $m_2 = m_1 - \bullet t + t\bullet$. For a sequence of transitions $\sigma = (t_1, \dots, t_n)$ we write $m_1 \xrightarrow{\sigma}_N m_{n+1}$, if $m_1 \xrightarrow{t_1}_N m_2 \xrightarrow{t_2}_N \dots \xrightarrow{t_n}_N m_{n+1}$, and we write $m_1 \xrightarrow{*}_N m_{n+1}$, if there exists such a sequence $\sigma \in T^*$. We will write $m_1 \xrightarrow{t} m_2$, $m_1 \xrightarrow{\sigma} m_{n+1}$ and $m_1 \xrightarrow{*} m_{n+1}$, if N is clear from the context.

We now introduce the notion of Workflow net, which is a Petri net where certain places and transitions are marked as input and output nodes. An *I/O net* is a tuple $N = (P, T, F, I, O)$ where (P, T, F) is a Petri net with a non-empty set $I \subseteq P \cup T$ of *input places* and a non-empty set $O \subseteq P \cup T$ of *output places*. In our setting we will restrict ourselves to I/O nets where input and output nodes are either all places, or all transitions. We will call such nets *I/O consistent*. As is usual for Petri nets that model workflows, we will also require that all nodes in the net can be reached from an input node, and from all nodes in the net an output node can be reached and call such nets *well-connected*.

Definition 1. (Workflow net)

A *Workflow net* (WF net) is an I/O net $N = (P, T, F, I, O)$ that is I/O consistent and well-connected.

The *I/O type* of a WF net is the type of its input and output nodes, i.e., it is *place* if it is pWF net, and *transition* if it is a tWF net.

Note that input places can have incoming edges in a workflow net, and that output places can have outgoing edges. We will refer to the nodes in $I \cup O$ as the *interface nodes* of the net. We will call a workflow net a *one-input* workflow net if I contains one element, and a *one-output* workflow net if O contains one element. Often, as in [27], workflow nets are restricted to one-input one-output place workflow nets. We generalise this in two ways: first by allowing also nets with input and output transitions rather than input and output places, and second by allowing multiple input and output places/transitions. For these generalised workflow nets we define the corresponding one-input one-output pWF net as follows. The *place-completion* of a tWF net $N = (P, T, F, I, O)$ is denoted as $\mathbf{pc}(N)$ and is a one-input one-output pWF net that is constructed from N by adding places p_i and p_o such that $p_i\bullet = I$ and $\bullet p_o = O$ and setting the input set and output set as $\{p_i\}$ and $\{p_o\}$, respectively. This is illustrated in Figure 1 (a). In such diagrams we will indicate nodes in I with an unconnected incoming arrow and nodes in O with an unconnected outgoing arrow. The *transition-completion* of a pWF net $N = (P, T, F, I, O)$ is denoted as $\mathbf{tc}(N)$ and is a one-input one-output tWF net that is constructed from N by adding transitions t_i and t_o such that $t_i\bullet = I$ and $\bullet t_o = O$ and setting the input set and output set as $\{t_i\}$ and $\{t_o\}$, respectively.

This is illustrated in Figure 1 (b).

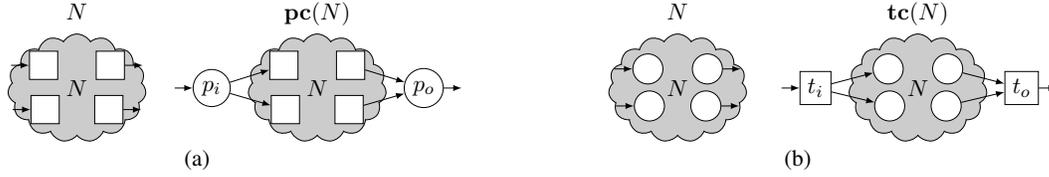


Figure 1. A place-completed tWF net and transition-completion pWF net

In this paper we discuss a particular kind of soundness, namely the soundness that guarantees the reachability of a proper final state [23]. We generalise this for the case where (1) there can be more than one input place and (2) these contain one or more tokens in the initial marking. We also provide a generalisation of soundness for tWF nets, which intuitively states that after k firings of all input transitions the computation can end in an empty marking while firing each of the output transitions exactly k times.

Definition 2. (k and *-soundness)

A pWF net $N = (P, T, F, I, O)$ is said to be k -sound if for each marking m such that $k \cdot I \xrightarrow{*} m$ it holds that $m \xrightarrow{*} k \cdot O$. We call N *-sound if it is k -sound for all $k \geq 1$. We say that these properties hold for tWF net N if they hold for $\text{pc}(N)$.

By definition place-completion does not affect the *-soundness. However, as we observed in [23, 22], for transition-completion this is only true in one direction as every pWF net N is *-sound if $\text{tc}(N)$ is *-sound but not vice versa.

3. Definition of AND-OR nets

In this section we introduce the AND-OR nets by recalling some definitions from [23, 22]. These are based on the notions of place substitution and transition substitution, where a place is replaced with a pWF net, and a transition is replaced with a tWF net.

Definition 3. (Place substitution, Transition substitution)

Consider two disjoint WF nets N and M , i.e., if $N = (P, T, F, I, O)$ and $M = (P', T', F', I', O')$, then $(P \cup T) \cap (P' \cup T') = \emptyset$. We then define *substitution* as follows. If p is a place in N , and M is a pWF net, then $N \otimes_p M$ is the result of substituting p in N with M , which is defined as follows:

- The node p is removed, and replaced with the nodes and edges of M .
- Each incoming edge (m, p) of p is removed, and replaced with the edges (m, p') for each $p' \in I'$.
- Each outgoing edge (p, m) of p is removed, and replaced with the edges (p', m) for each $p' \in O'$.
- If $p \in I$ then it is removed from I and all $p' \in I'$ are added to I .
- If $p \in O$ then it is removed from O and all $p' \in O'$ are added to I .

Analogously, *transition substitution* is defined when t is a transition in N and M is a tWF net, in which case $N \otimes_t M$ is the result of substituting t in N with M defined analogously.

The results of a place substitution and transition substitution are illustrated in Figure 2 (a) and (b), respectively. It is not hard to see that for all WF nets N and M and n a node in N such that $N \otimes_n M$ is defined, it is again a WF net. It also holds for all WF nets A , B and C that $(A \otimes_n B) \otimes_m C = A \otimes_n (B \otimes_m C)$, and $(A \otimes_n B) \otimes_m C = (A \otimes_n C) \otimes_m B$ if n and m are different nodes in A .

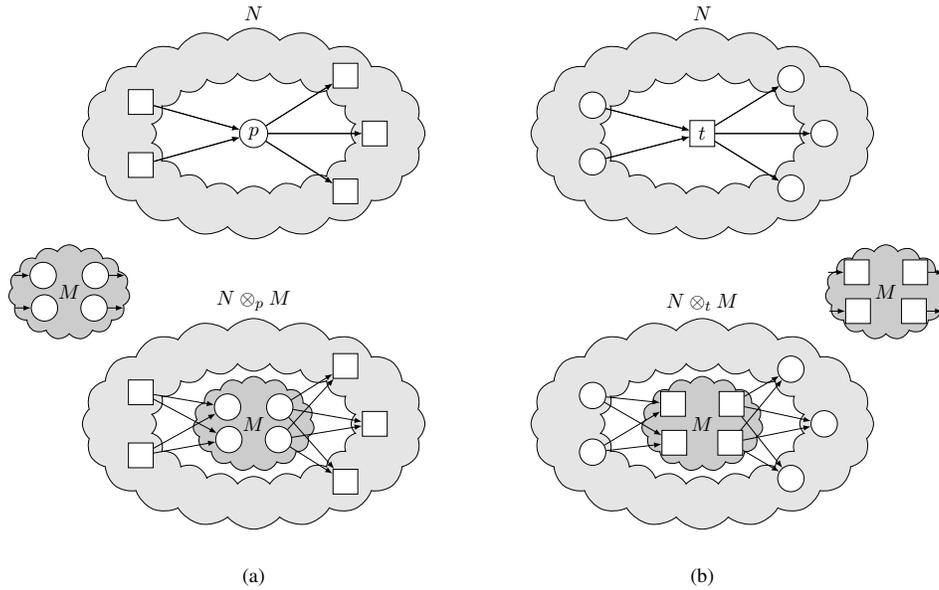


Figure 2. Illustration of place substitution and transition substitution (adapted from [23])

We will generate nets by starting from some basic classes C of nets and allowing substitutions of places with pWF nets and transitions with tWF nets in nets in this class. For that we use the notion of *substitution closure* of C denoted as $\mathbf{S}(C)$ and defined as the closure of C under transition substitution and place substitution, i.e., the smallest superclass $\mathbf{S}(C)$ of C that satisfies the following two rules for every two disjoint nets N and M in $\mathbf{S}(C)$: (1) if M is a pWF net and p a place in N then $N \otimes_p M$ is a net in $\mathbf{S}(C)$ and (2) if M is a tWF net and t a transition in N then $N \otimes_t M$ is a net in $\mathbf{S}(C)$.

The motivation for using the concept of substitution closure is that we can prove certain properties of these nets, such as a certain kind of soundness, by showing that these properties (1) hold for the initial class of nets and (2) are preserved by substitution. As was shown in earlier work [23], *-soundness is not always preserved by substitution, but substitution does preserve a stronger property called *substitution soundness*. The intuition behind this notion of soundness is that the execution of a net, when started with the same number of tokens in all input places, should always be able to finish properly, even if the execution is interfered with by removing at some step an identical number of tokens from all output places. More precisely, if we start the net with k tokens in all its input places, and somewhere during the execution remove k' tokens from each of the output places, then the net should be able to finish with $k - k'$ tokens in each of its output places.

Definition 4. (Substitution soundness)

A pWF net $N = (P, T, F, I, O)$ is said to be *substitution sound* if for all $0 \leq k' \leq k$ and every marking m' of N such that $k.I \xrightarrow{*} m' + k'.O$ it holds that $m' \xrightarrow{*} (k - k').O$. A tWF net is said to be substitution sound if its place-completion is substitution sound.

It is easy to see that substitution soundness implies *-soundness, since *-soundness only considers the cases where $k' = 0$, and so every substitution sound net is also a *-sound net.

The choice for the initial classes is made such that these are indeed substitution sound. They are based on acyclic marked graphs (T-nets) and state machines (S-nets), as considered in [32]. The basic idea of T-nets is that during executions no choices are made about who consumes a token and all transitions fire in the execution of a workflow. This is captured by the syntactic restriction that places have exactly one incoming edge and one outgoing edge, where being an input node or output node counts as an additional incoming edge or outgoing edge, respectively. We will call this the *AND-property* of a WF net.

Definition 5. (AND property)

A WF net $N = (P, T, F, I, O)$ is said to have the *AND property* if for every place $p \in P$ it holds that (1) $p \in I \wedge |\bullet p| = 0$ or $p \notin I \wedge |\bullet p| = 1$ and (2) $p \in O \wedge |p \bullet| = 0$ or $p \notin O \wedge |p \bullet| = 1$.

The basic idea of S-nets is that they represent a state machine, with the state represented by a single token that is in one of the places. This is captured by a restriction that says that transitions have exactly one incoming edge and one outgoing edge, where being an input node or output node counts as an additional incoming edge or outgoing edge, respectively. We will call this the *OR-property* of a net.

Definition 6. (OR property)

A WF net $N = (P, T, F, I, O)$ is said to have the *OR property* if for every transition $t \in T$ it holds that (1) $t \in I \wedge |\bullet t| = 0$ or $t \notin I \wedge |\bullet t| = 1$ and (2) $t \in O \wedge |t \bullet| = 0$ or $t \notin O \wedge |t \bullet| = 1$.

Unfortunately not all WF nets with the AND property or the OR property are substitution sound. Consider for example a net with the AND property containing a cycle of transitions and places. All places in this cycle will have no other incoming and leaving edges than those of the loop. In such a net the transitions in the cycle can never fire, since that would require a token in their preceding place, but such a token can only be generated by firing a transition in the cycle. To remedy this, we will define AND nets as WF nets that not only satisfy the AND property but are also acyclic, which leads to the following definitions of AND and OR nets.

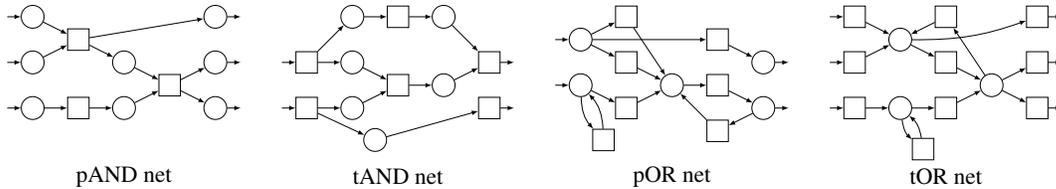


Figure 3. Examples of a pAND, tAND, pOR and tOR nets (adapted from [23])

Definition 7. (AND net)

An *AND net* is an acyclic WF net that satisfies the AND property. An AND net that is a pWF net is called a pAND net, and if it is a tWF net it is called a tAND net.

Definition 8. (OR net)

An *OR net* is a WF net that satisfies the OR property. An OR net that is a pWF net is called a pOR net, and if it is a tWF net it is called a tOR net.

For some examples of AND and OR nets see Figure 3. The classes of pOR nets, pAND nets, tOR and tAND nets will be referred to as **pOR**, **pAND**, **tOR** and **tAND**. In addition we will prefix the name with **11** if the class contains only nets with one input node and one output node.

The purpose of the AND and the OR nets is to define an initial class of substitution sound nets. Unfortunately, although all pAND and tOR nets are substitution sound, this is not the case for all tAND and pOR net. The cause of this is illustrated by the tAND net in Figure 3. Recall that the soundness of a tWF net is defined as the soundness of its place-completion which adds a single place before the input transitions and a single place after the output transitions. So if we put a single token in the first place in this place-completion, then only one of the first transitions can fire, after which the net can no longer correctly finish. A similar problem occurs in the pOR net in Figure 3. If it starts with a token in each of its input places, then it cannot finish correctly if it moves the token of the upper input place to the place in the middle of the net. Both types of problems are solved if we only consider one-input one-output versions of tAND and pOR nets, which can all be shown to be substitution sound nets. We will refer to these classes of nets as **11tAND** and **11pOR**, respectively.

Given the previous considerations, we will for the purpose of generating substitution sound nets only consider the classes **pAND**, **11tAND**, **11pOR** and **tOR**. We will refer to these classes as the *basic AND-OR classes*, and to the class that is obtained by combining them by substitution as the class of *AND-OR nets*.

Definition 9. (AND-OR net)

The substitution closure $S(\mathbf{pAND} \cup \mathbf{11tAND} \cup \mathbf{11pOR} \cup \mathbf{tOR})$ is called the class of AND-OR nets.

An example of the generation of an AND-OR net is shown in Figure 4, with on the left the hierarchical decomposition and on the right the resulting net.

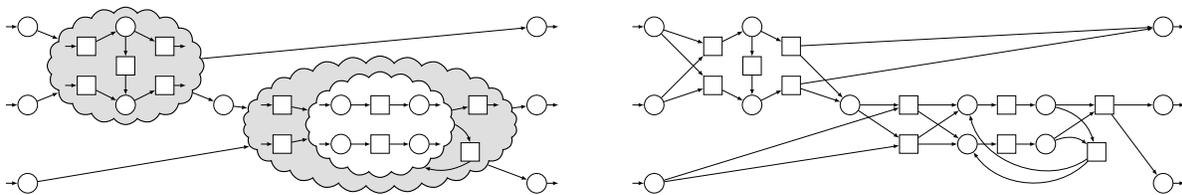


Figure 4. An example of the generation of an AND-OR net (adapted from [23])

4. The AND-OR reduction

We now proceed with presenting the AND-OR net verification procedure. Informally, the procedure can be described as reversing the generation process. This means that we try to find subnets that might have been generated by a substitution and reverse the substitution. This process is repeated until we can find no more such subnets, and if then the resulting net is a single node, the original net is concluded to be an AND-OR net.

Definition 10. (Subnet)

A subnet in a larger workflow net $M = (P_M, T_M, F_M, I_M, O_M)$ is identified by a non-empty set of nodes $S \subseteq P_M \cup T_M$. With S we associate a net $M[S] = (P_S, T_S, F_S, I_S, O_S)$ that is the restriction of M to the nodes in S , i.e., (1) $P_S = P_M \cap S$, (2) $T_S = T_M \cap S$ and (3) $F_S = F_M \cap (S \times S)$. Moreover, the input nodes of $M[S]$ are the nodes in S that are input nodes of M plus the nodes in S that have in M incoming edges from outside S , i.e., (4) $I_S = (I_M \cap S) \cup \{n_2 \mid (n_1, n_2) \in F_M, n_1 \in ((P_M \cup T_M) \setminus S), n_2 \in S\}$ and analogously (5) $O_S = (O_M \cap S) \cup \{n_1 \mid (n_1, n_2) \in F_M, n_1 \in S, n_2 \in ((P_M \cup T_M) \setminus S)\}$. A subnet consisting of exactly one node will be called a *trivial subnet*.

Not every subnet of a WF net is again itself a WF net, but it is straightforward to show that it will always be a well-connected I/O net, which follows from the way how input and output nodes are defined. It then follows that a subnet of WF net is itself a WF net iff it is I/O consistent.

We now proceed with defining the notion of *contraction*, i.e., a contraction of a subnet S of a WF net M into a single new node n .

Definition 11. (Contraction)

Given a WF net $M = (P, T, F, I, O)$ and a subnet S of M such that $M[S]$ is a pWF net, we define the result of contracting S into a place node $n \notin (P \cup T) \setminus S$ as the I/O net M' which is identical to M except that (1) all places and transitions in S and the edges between them are replaced with the single place n , (2) all edges (n_1, n_2) in M from a node not in S to a node in S are replaced with an edge (n_1, n) , (3) all edges (n_1, n_2) in M from a node in S to a node outside S are replaced with (n, n_2) , (4) if any node in S is an input node of M then n is an input node of M' and (4) if any node in S is an output node of M then n is an output node of M' . The result of contracting a subnet S such that $M[S]$ defines a tWF is the same except that the new node n is a transition.

Observe that the result of a contraction is indeed a Petri net. This is because, if a new edge in M' connects n to an old node n' then there must have been an edge in M between n' and an input or output node of $M[S]$. It follows that n' is a transition if n is a place, and a place if n is a transition.

It is also not hard to see that M' is I/O consistent and that the type of its input (and output) nodes will be the same as that of M . After all, the new node n becomes only an input (output) node if one of the nodes in S is an input (output) node of M . This node in S , say n' , is by definition also an input (output) node of $M[S]$, and so a transition if $M[S]$ is a tWF net and a place if it is a pWF net. In the first case n is a transition, and so of the same type as n' , and in the second case n is a place, and so then also of the same type as n' . Since M' is I/O consistent, it follows that n is of the same type as all the other input (output) nodes of M' , which were already input (output) nodes of M .

Like for the usual notion of contraction in graph theory, it is clear that paths are preserved, i.e., if there was a path between two nodes in a WF net then there will still be a similar path in the result of a contraction where all maximal substrings of nodes in the contracted subnet are replaced with the new

node. This ensures that the result is again well-connected, and together with the observations that the net remains a Petri net and that the types of the input and output nodes will remain the same, it follows that the result of a contraction in a pWF (tWF) net is again a pWF (tWF) net.

During the reduction we will search for non-empty subsets of nodes S in a workflow net M such that their associated net $M[S]$ is a non-trivial basic AND-OR net, and contract them into a single new node. Such a contraction should be the inverse of a substitution of this new node with $M[S]$ in the sense that if we apply the contraction, and then the substitution, we should again have the same WF net. However, applying that substitution may not always give back M , as is illustrated in Figure 5 (a) and (b). If in a WF net (a) we first contract the net indicated by the cloud and identified by $S = \{p_1, p_2, t_3, p_3, p_4\}$, and then substitute the contracted net again, we obtain the net shown in (b). The difference can be understood by considering the *external preset* and *external postset* of nodes in the contracted net, i.e., the subset of the preset not in S and the subset of the postset not in S . After a substitution it always holds that all input nodes of the substituted net of the same external preset. For example, in (b) the nodes p_1 and p_2 have both $\{t_1, t_2\}$ as their external preset. Similarly, all output nodes of the substituted net will all have the same external postset. For example, in (b) the nodes p_3 and p_4 have both the external postset $\{t_4, t_5\}$. We will call this property concerning the external presets and external postsets the *well-nestedness* of a subnet. It is easily observed that the subnet in (a) does not satisfy it since the external presets of p_1 and p_2 are $\{t_1\}$ and $\{t_1, t_2\}$, respectively, and in addition the external postsets of p_3 and p_4 are $\{t_4\}$ and $\{t_5\}$. It is not hard to see that well-nestedness of a subnet is both a necessary and sufficient requirement for a substitution to undo a contraction. We now proceed with the formal definition of the well-nestedness condition.

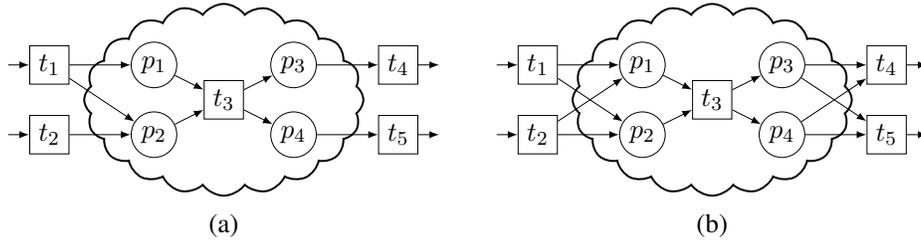


Figure 5. Example of a non-well-nested subnet (a) and a well-nested subnet (b)

Definition 12. (Well-nested subnet)

A subnet S is said to be well-nested in M (or for short $M[S]$ is well-nested) if it holds that (1) for any two input nodes n_1, n_2 in $M[S]$ we have $\bullet_M n_1 \setminus S = \bullet_M n_2 \setminus S$ and (2) for any two output nodes n_1, n_2 in $M[S]$ we have $n_1 \bullet_M \setminus S = n_2 \bullet_M \setminus S$.

A subnet that is well-nested, and where the associated net $M[S]$ is WF/pWF/tWF net, will be called a *well-nested WF/pWF/tWF net*.

Definition 13. (Contractible)

We will call a subnet of M *contractible* if it is (1) a basic AND-OR net, (2) well-nested in M and (3) non-trivial.

Recall from Section 3 that the basic AND-OR classes are pAND, 11tAND, 11pOR and tOR. Based on this we define the contraction relation which will form the basis of the reduction process.

Definition 14. (Contraction relation)

The *contraction relation* is the binary relation \rightsquigarrow over WF nets such that $M \rightsquigarrow M'$ expresses that (1) there is a contractible WF net N in M and (2) the contraction of N in M results in M' . The reflexive and transitive closure of \rightsquigarrow is denoted as \rightsquigarrow^* .

Based on this, we now define the notion of an AND-OR reduction of a WF net, in this paper simply referred to as a reduction of a WF net, which describes the result of reducing a WF net by applying contractions until no more such contractions are possible.

Definition 15. (Reduction)

A *reduction* of a WF net M is a WF net M' such that (1) $M \rightsquigarrow^* M'$ and (2) there is no WF net M'' such that $M' \rightsquigarrow M''$.

The main result of this paper is that the reduction of a WF net is uniquely defined up to isomorphism, i.e., all results are identical up to the choice of the identity of the nodes. We write $M \sim M'$ to indicate that M and M' are isomorphic, and $[M]$ to denote the equivalence class of all WF nets isomorphic to M . The main theorem then can be formulated as follows:

Theorem 16. (Unique result up to isomorphism of the reduction)

Let M_1 and M_2 be both reductions of a WF net M , then $M_1 \sim M_2$.

The proof of this theorem will be based on showing that the reduction procedure is locally confluent up to isomorphism, and then showing that from this we can conclude that the reduction procedure is globally confluent up to isomorphism. We first state the theorem describing local confluence up to isomorphism.

Theorem 17. (Local confluence up to isomorphism of the contraction relation)

For all WF nets M , M_1 and M_2 it holds that if $M \rightsquigarrow M_1$ and $M \rightsquigarrow M_2$, then there are isomorphic WF nets M_3 and M_4 such that $M_1 \rightsquigarrow^* M_3$ and $M_2 \rightsquigarrow^* M_4$.

Since the proof of this theorem is rather involved, we will describe it separately in Section 5. Based on this theorem we can then give the proof of Theorem 16:

Proof:

(of Theorem 16) Based on the contraction relation we can define a *class-based contraction relation* over equivalence classes of WF nets such that $[M_1] \Rightarrow [M_2]$ iff for some $M'_1 \in [M_1]$ and some $M'_2 \in [M_2]$ it holds that $M'_1 \rightsquigarrow M'_2$. Let a *class-based reduction* of a class $[M]$ then be a class $[M']$ such that (1) $[M] \Rightarrow^* [M']$ and (2) there is no WF net M'' such that $[M'] \Rightarrow [M'']$.

We can first make the observation that contractions respect isomorphisms in the sense that if $M \rightsquigarrow M_1$ and M' and M are isomorphic then there is an M_2 isomorphic to M_1 such $M' \rightsquigarrow M_2$. Because of this, it follows from the local confluence up to isomorphism for \rightsquigarrow , as stated in Theorem 17, that there is local confluence for \Rightarrow , i.e., for all WF nets M , M_1 and M_2 it holds that if $[M] \Rightarrow [M_1]$ and $[M] \Rightarrow [M_2]$, then there is a WF net M_3 such that $[M_1] \Rightarrow^* [M_3]$ and $[M_2] \Rightarrow^* [M_3]$. It is clear that the class-based contraction relation is well-founded, since each contraction reduces the number of nodes

with at least one. So it follows by *Newman's Lemma* (sometimes also called the *diamond Lemma*, see [12]) that \Rightarrow is globally confluent, i.e., the class-based reduction of a WF net class is unique.

Now, assume that M_1 and M_2 are both reductions of a WF net M . Then it holds that (1) $M \rightsquigarrow^* M_1$ and there is no M_3 such that $M_1 \rightsquigarrow M_3$ and (2) $M \rightsquigarrow^* M_2$ and there is no M_4 such that $M_1 \rightsquigarrow M_4$. By induction on the number of contractions and the definition of \Rightarrow , it follows that (1) $[M] \Rightarrow^* [M_1]$ and there is no M_3 such that $[M_1] \Rightarrow [M_3]$ and (2) $[M] \Rightarrow^* [M_2]$ and there is no M_4 such that $[M_1] \Rightarrow [M_4]$. Therefore $[M_1]$ and $[M_2]$ are class-based reductions of $[M]$, and since \Rightarrow is globally confluent it follows that $[M_1] = [M_2]$, or in other words $M_1 \sim M_2$. \square

5. Local confluence up to isomorphism of contractions

In this section we show the local confluence up to isomorphism of the contraction relation as stated by Theorem 17. To simplify the presentation of the proof, we will at first restrict ourselves to contractions that do not involve input and output nodes of the larger net, i.e., where $S \cap I_M = S \cap O_M = \emptyset$. It is easy to see that for such contractions the sets of input and output nodes of the larger net do not change, i.e., if M' is the result of the contraction then $I_{M'} = I_M$ and $O_{M'} = O_M$, which simplifies the definition of the result of the contraction. We will call such contractions *internal contractions*, and a well-nested WF net that does not share input nor output nodes with the larger WF net will be called an *internal well-nested WF net*. We will call an internal well-nested non-trivial net that is of one of the basic AND-OR classes *internal contractible*. We let $\rightsquigarrow_{\text{int}}$ denote the restriction of \rightsquigarrow where only internal contractions are used.

The restriction to internal contractions is interesting because the general reduction process can be simulated by the one restricted to internal contractions. More precisely, there is a generation contraction from M to M' iff there is an internal contraction from the completion of M to the completion of M' . It can then be shown with induction that a WF net M' is the result of a general reduction applied to M iff the completion of M' is the result of a reduction with only internal contractions starting from the completion of M .

Since in the remainder of this section we will talk mostly about internal contractions, we will refer to internal contractions, internal well-nested subnets, and internal contractible nets simply as *contractions*, *well-nested subnets* and *contractible nets*, respectively, unless stated otherwise.

The basic idea of the proof for local confluence is roughly that if there are two contractible subnets then (1) we can contract them in any order, i.e., after contracting one net the other remains contractible, and (2) regardless of the contraction order we obtain the same result up to isomorphism. This is clearly true if the two nets are separated as defined by the flow relation, but becomes more complicated to see if they are connected by some edges or even overlap. For the latter possibilities we distinguish four cases which are illustrated in Figure 6. In this figure we use clouds such as S_1 and S_2 to indicate subnets and rounded rectangles such as n_1 and n_2 to represent nodes that can be either places or transitions. The cases we distinguish are as follows:

- (A) *The subnets S_1 and S_2 do not share nodes, but nodes in S_1 might be connected to nodes in S_2 :* In this case the contraction of one subnet will not change the other subnet, but it might change the edges connected to it. However, as will be shown, this subnet will remain well-nested and therefore contractible. Note that there might be multiple edges between S_1 and S_2 , but after contracting the subnets defined by them into n_1 and n_2 , there will be at most one edge from n_1 to n_2 , and vice versa.

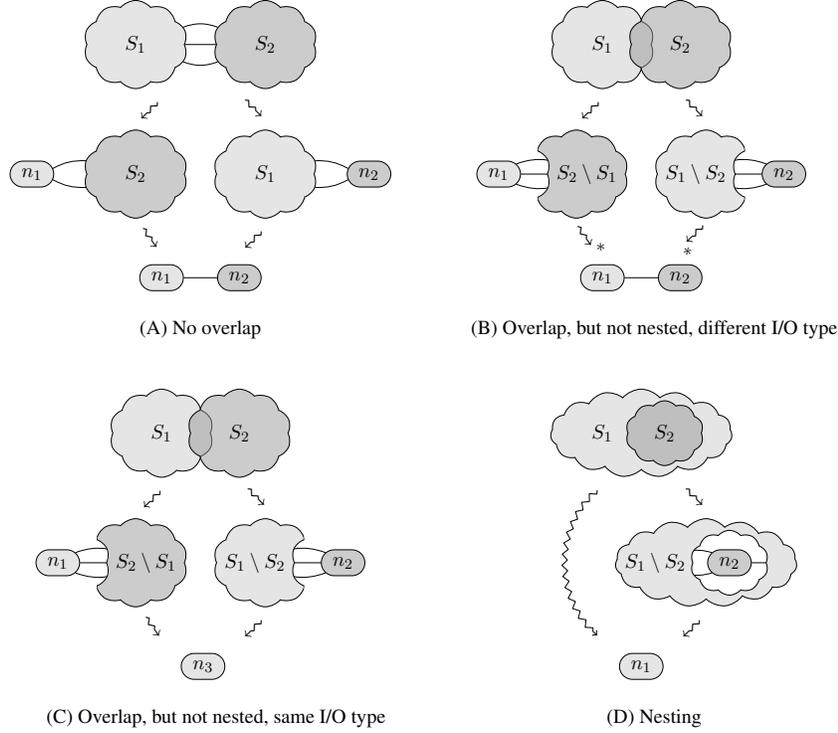


Figure 6. The four cases considered in the confluence proof

- (B) *The subnets S_1 and S_2 overlap, but are not nested in one another and the I/O type of the two nets is different:* In this case the contraction of one subnet, say S_1 , removes a part of S_2 . It must therefore be shown that after the contraction the remainder of S_2 , i.e., the subnet $S_2 \setminus S_1$ is either a trivial net, in which case no further contraction is necessary, or a contractible net, i.e., $M[S_2 \setminus S_1]$ is non-trivial, well-nested and belongs to a basic AND-OR class. In fact, it will be shown that it will belong to the same basic AND-OR class as $M[S_2]$. Note that since $M[S_1]$ and $M[S_2]$ have different I/O types, it follows that n_1 and n_2 have different types, which is required in a Petri net for connected nodes. Also note that In the case where $S_2 \setminus S_1$ is a trivial net, and therefore non-contractible, the node n_2 in the final result is the one node in $S_2 \setminus S_1$ and not necessarily the node that S_2 is contracted into in the right-hand sequence of contractions. However, the final results of both sequences contractions will be isomorphic. It is because of this that we only have here confluence up to isomorphism and not actual confluence.

It must also be shown that the final result with nodes n_1 and n_2 is the same up to isomorphism, independent of whether first S_1 or first S_2 is contracted. This means that (a) the edges between n_1 and n_2 must be the same, (b) the edges between n_1 and nodes other than n_2 must be the same and (c) the edges between n_2 and nodes other than n_1 must be the same.

- (C) *The subnets S_1 and S_2 overlap, are not nested in one another and the I/O type of the two nets is the same:* In this case we contract in the second step not just the subnet consisting of the remainder of

the subnet that was not contracted, say $M'[S_2 \setminus S_1]$ where M' is the result of the first contraction, but the total of that remainder plus the new node, which would be $M'[(S_2 \setminus S_1) \cup \{n_1\}]$. Therefore it must be shown that this subnet is a contractible net. In fact, it will be shown that it will belong to the same basic AND-OR class as $M[S_2]$. Note that since $M[S_1]$ and $M[S_2]$ have the same I/O type, we cannot simply contract $S_2 \setminus S_1$ in M' , since that would result in a node of the same type as n_1 .

It must also be shown that the final result with node n_3 is the same, independent of whether first S_1 or first S_2 is contracted. This means that n_3 must be participating in the same edges.

- (D) *The subnet S_1 contains the subnet S_2 :* The first option is to contract S_1 into n_1 , in which case S_2 completely disappears. The second option is to contract S_2 into n_2 . It will be shown that after this step the subnet defined by the remainder of S_1 plus the new node n_2 , i.e., $(S_2 \setminus S_1) \cup \{n_2\}$, is contractible in the net resulting from the first contraction. In fact, it will be shown that it will belong to the same basic AND-OR class as $M[S_1]$ in the original workflow net.

It must also be shown that the final result with node n_1 is the same, independent of whether first S_1 or first S_2 is contracted. This means that n_1 must be participating in the same edges.

In the following lemmas we will discuss the necessary proofs for the preservation of contractibility of the non-contracted subnet for each of the mentioned cases in detail.

Lemma 18. (Case (A): preservation of contractibility)

Let M be a WF net with two contractible subnets S_1 and S_2 such that $S_1 \cap S_2 = \emptyset$. Then, if M' is the result of contracting $M[S_1]$ into n_1 then S_2 is contractible in M' .

Proof:

All requirements for contractibility will stay valid for S_2 , since its nodes and edges are not changed by the reduction, except that the arriving and departing edges from outside S_2 might have changed. So we need to verify if S_2 remains well-nested. Since we are only considering internal contractions, it is sufficient to show that in M' for every input node n of $M'[S_2]$ the set $\bullet_{M'n} \setminus S_2$ is the same, and that for every output node n of $M'[S_2]$ the set $n \bullet_{M'} \setminus S_2$ is the same. Since the contraction of S_1 simply replaces in both these sets the nodes from S_1 with n_1 , it follows that this indeed still holds in M' . \square

Lemma 19. (Case (B): preservation of contractibility)

Let M be a WF net with two contractible subnets S_1 and S_2 such that $S_1 \cap S_2 \neq \emptyset$, $S_1 \not\subseteq S_2$, $S_2 \not\subseteq S_1$ and $M[S_1]$ and $M[S_2]$ have different I/O types. Then, if M' is the result of contracting S_1 into n_1 then in M' the subnet $S_2 \setminus S_1$ is either contractible or a trivial subnet.

Proof:

We need to show that $M'[S_2 \setminus S_1]$ is (a) well-nested and (b) a WF net of a basic AND-OR type. For (a) the argument is as follows. We first consider the input nodes of $M'[S_2 \setminus S_1]$, for which we need to show that for each such node n' the set $\bullet_{M'n'} \setminus (S_2 \setminus S_1)$ is the same. If all the input nodes of $M'[S_2 \setminus S_1]$ are also input nodes of $M[S_2]$, this is clearly the case since $\bullet_{M'n'}$ is equal to $\bullet_{Mn'}$ except that each node from S_1 is replaced with n . If $M'[S_2 \setminus S_1]$ has an input node n'' that is not also an input node of $M[S_2]$ then for all input nodes n' of $M'[S_2 \setminus S_1]$ it can be shown that $\bullet_{M'n'} \setminus (S_2 \setminus S_1) = \{n\}$ as follows.

It can be observed that all output nodes of $M[S_1]$ are in $S_1 \cap S_2$. Otherwise, if there was an output node in $S_1 \setminus S_2$, it would, because of the well-nestedness of S_1 in M , have in M an outgoing edge to

n'' , and so n'' would already have been an input node of $M[S_2]$. From this observation and the well-connectedness of S_1 in M we can conclude that there must be in M an edge from a node, say n_3 , in $S_1 \setminus S_2$ to a node, say n_4 , in $S_1 \cap S_2$. It then follows that n_4 is an input node of $M[S_2]$. We can then show by contradiction that $\bullet_{M'} n' \setminus (S_2 \setminus S_1)$ only contains node n . Assume that it contains another node $n_5 \notin S_1 \cup S_2$. Since n_4 is also an input node of $M[S_2]$ and $M[S_2]$, it follows that n_4 has an incoming edge from n_5 . But this would make n_4 both an input node of $M[S_1]$ and of $M[S_2]$, which contradicts the assumption that $M[S_1]$ and $M[S_2]$ have different I/O types.

The proof for well-nestedness of the output nodes of $M'[S_2 \setminus S_1]$ is similar to that for the input nodes, except that the direction of the edges is reversed and the set of input nodes is replaced with the set of output nodes.

We now proceed with proof of statement (b), which states that $M'[S_2 \setminus S_1]$ is a WF net of a basic AND-OR type. We will show that (1) its input and output nodes have the same types as those of $M[S_2]$, (2) it has the AND (OR) property if $M[S_2]$ has it, (3) it is acyclic if $M[S_2]$ is acyclic and (4) it has the one-input-one-output property if $M[S_2]$ is a 1IpOR or 1ItAND net. We consider each of these in the following.

- (1) **Preservation of I/O type:** We first consider input nodes. Clearly if all input nodes of $M'[S_2 \setminus S_1]$ were already input nodes of $M[S_2]$ their type in $M'[S_2 \setminus S_1]$ is the same as in $M[S_2]$. For new input nodes it will hold they have in M' an incoming edge from n , and so have the opposite I/O type of $M[S_1]$, which in turn is by assumption the opposite of the I/O type of $M[S_2]$, so the new input node has as its type the I/O type of $M[S_2]$. The argument is similar for output nodes.
- (2) **Preservation of the AND property and the OR property:** We first consider the AND property, and specifically the restriction on input edges for places. For nodes that are not input nodes of $M'[S_2 \setminus S_1]$ this property is preserved from $M[S_2]$ since edges and nodes are only removed and not added. For the same reason the property is also preserved for input nodes of $M'[S_2 \setminus S_1]$ that in M' are not connected to n . Finally, for input nodes in $M'[S_2 \setminus S_1]$ that are connected to n , there must have been in M an edge from a node in $S_1 \cap S_2$ to that node. So if its incoming edges violate the AND property after the contraction, they would already have done so before the contraction.

The argument for outgoing edges is analogous, and also for the incoming and outgoing edges violating the OR property.

- (3) **Preservation of acyclicity:** If there is a cycle in $M'[S_2 \setminus S_1]$ then that cycle would also have existed in $M[S_2]$ since it contains a superset of the edges.
- (4) **Preservation of the one-input-one-output property:** We show that if $M[S_2]$ is a 1IpOR net or a 1ItAND net, then $M'[S_2 \setminus S_1]$ also has the one-input-one-output property. We show this by contradiction. We start with assuming that $M'[S_2 \setminus S_1]$ has at least two input nodes, say n_3 and n_4 , and then show that $M[S_2]$ cannot be a 1IpOR net nor a 1ItAND net. Clearly this holds if n_3 and n_4 were already input nodes of $M[S_2]$. For the case with one of the two nodes, say n_3 , an input node of $M[S_2]$, and the other n_4 not, we can argue as follows. The old input node n_3 must have in M an incoming edge from a node n_5 outside S_2 , and n_5 is either in S_1 or not. In the first case, it follows from the well-nestedness of S_1 in M and the fact that n_4 has in M an incoming edge from a node in S_1 that there is an edge (n_5, n_4) in M , which contradicts that n_4 is not an input node of $M[S_2]$. In the second case, where $n_5 \notin S_1$, it follows from the well-nestedness of $S_2 \setminus S_1$

in M' , as shown earlier in this proof, that there is an edge (n_5, n_4) , which also here contradicts that n_4 is not an input node of $M[S_2]$. Finally, we consider the case where both n_3 and n_4 are not input nodes in $M[S_2]$. Since both became input nodes in $M[S_2 \setminus S_1]$ and $M[S_1]$ is well-nested there must be a node n_5 in $S_1 \cap S_2$ and edges (n_5, n_3) and (n_5, n_4) in M . The node n_5 is either a place or a transition. If it is a place, both n_3 and n_4 are transitions and $M[S_2]$ violates the AND property because n_5 is a place with two outgoing edges, so $M[S_2]$ is not a 11tAND net. On the other hand, it is also not a 11pOR net, since n_5 is an output node of $M[S_1]$, which implies that $M[S_1]$ is a pWF net, and so $M[S_2]$ a tWF net. So we have a contradiction with the assumption that $M[S_2]$ is a 11pOR net or a 11tAND net. By an analogous argument it can be shown we get a similar contradiction if n_5 is a transition.

In a similar fashion it can be shown that if we assume $M'[S_2 \setminus S_1]$ has at least two output nodes, it then follows that $M[S_2]$ cannot be a 11pOR net nor a 11tAND net.

Note that from the preservation of the type of the input and output nodes, it follows that $M'[S_2 \setminus S_1]$ is I/O consistent, and since it is a subnet of a WF net it follows that it is itself a WF net. Since all the other properties that must hold for a WF net of a basic AND-OR type were also shown to be preserved, it follows that $M'[S_2 \setminus S_1]$ is a WF net of basic AND-OR type. This net will either contain only one node, in which case it is trivial, or contain more than one node, in which case it is contractible. \square

Lemma 20. (Case (C): preservation of contractibility)

Let M be a WF net with two contractible subnets S_1 and S_2 such that $S_1 \cap S_2 \neq \emptyset$, $S_1 \not\subseteq S_2$, $S_2 \not\subseteq S_1$ and $M[S_1]$ and $M[S_2]$ have the same I/O type. Then, if M' is the result of contracting S_1 into n_1 then in M' the subnet $(S_2 \setminus S_1) \cup \{n_1\}$ is a contractible subnet.

Proof:

For succinctness, we will write S'_2 for the set $(S_2 \setminus S_1) \cup \{n_1\}$. We need to show that (a) in M' the subnet S'_2 is well-nested and (b) $M'[S'_2]$ is a WF net of a basic AND-OR type. For (a) the argument is as follows. We first consider the input nodes of $M'[S'_2]$, for which we need to show that for each such n' the set $\bullet_{M'} n' \setminus S'_2$ is the same. As in the proof for case (B) this is clear if all the input nodes of $M'[S'_2]$ are also input nodes of $M[S_2]$. If $M'[S'_2]$ has an input node n'' that is not also an input node of $M[S_2]$ then $n'' = n_1$, since none of the nodes in S_1 get in M' new incoming edges except from n_1 . So it remains to verify that $\bullet_{M'} n' \setminus S'_2 = \bullet_{M'} n_1 \setminus S'_2$ for all input nodes n' of $M'[S'_2]$, which can be shown as follows.

Clearly this holds for $n' = n_1$, so we assume that $n' \in S_2 \setminus S_1$. It can now be observed that not all input nodes of $M[S_1]$ are in $S_1 \setminus S_2$. We show this by contradiction, and start with assuming that all input nodes of $M[S_1]$ are in $S_1 \setminus S_2$. Given the well-connectedness of S_1 and the non-emptiness of $S_1 \setminus S_2$, there must be an edge in M from a node in $S_1 \setminus S_2$ to a node, say n_2 , in $S_1 \cap S_2$. Since n' is an input node of $M'[S'_2]$, there is an edge (n_3, n') in M with $n_3 \notin S_1 \cup S_2$. As $n' \in S_2 \setminus S_1$ it is also an input node of $M[S_2]$. Because $M[S_2]$ is well-nested, it follows there is also an edge (n_3, n_2) in M . But this means that n_2 is an input node of $M[S_1]$, which contradicts the assumption that all input nodes of $M[S_1]$ are in $S_1 \setminus S_2$.

Given the previous observation, we can assume that there is an input node n_4 of $M[S_1]$ in $S_1 \cap S_2$. Since S_1 is contracted into n_1 it follows from the semantics of contraction that $\bullet_{M'} n_4 \setminus S_1 = \bullet_{M'} n_1 \setminus \{n_1\}$. From this it follows that $\bullet_{M'} n_4 \setminus (S_1 \cup S_2) = \bullet_{M'} n_1 \setminus (\{n_1\} \cup S_2)$. Since nodes from S_1 no longer appear in M' , we get (i) $\bullet_{M'} n_4 \setminus (S_1 \cup S_2) = \bullet_{M'} n_1 \setminus ((S_2 \setminus S_1) \cup \{n_1\}) = \bullet_{M'} n_1 \setminus S'_2$. Since

$M[S_2]$ and n' and n_4 are both input nodes, $\bullet_{Mn_4} \setminus S_2 = \bullet_{Mn'} \setminus S_2$. From this it follows that (ii) $\bullet_{Mn_4} \setminus (S_1 \cup S_2) = \bullet_{Mn'} \setminus (S_1 \cup S_2)$. Because $n' \notin S_1$ it follows from the semantics of contraction that $\bullet_{Mn'} \setminus S_1 = \bullet_{M'n'} \setminus \{n_1\}$. It follows that $\bullet_{Mn'} \setminus (S_1 \cup S_2) = \bullet_{M'n'} \setminus (\{n_1\} \cup S_2)$. Since nodes from S_1 no long appear in M' , we can conclude (iii) $\bullet_{Mn'} \setminus (S_1 \cup S_2) = \bullet_{M'n'} \setminus ((S_2 \setminus S_1) \cup \{n_1\}) = \bullet_{M'n'} \setminus S'_2$. By combining (i), (ii) and (iii) we get that $\bullet_{M'n_1} \setminus S'_2 = \bullet_{Mn_4} \setminus (S_1 \cup S_2) = \bullet_{Mn'} \setminus (S_1 \cup S_2) = \bullet_{M'n'} \setminus S'_2$

The proof for well-nestedness of the output nodes of $M'[S_2 \setminus S_1]$ follows by analogy.

We now proceed with proof of statement (b) that is $M'[S'_2]$ is a WF net of a basic AND-OR type. We will show that (1) its input and output nodes have the same types as those of $M[S_2]$, (2) it has the AND (OR) property if $M[S_2]$ has it, (3) it is acyclic if $M[S_2]$ is a pAND net or a 11tAND net and (4) it has the one-input-one-output property if $M[S_2]$ has it. We consider each of these in the following.

- (1) **Preservation of I/O type:** We first consider input nodes. The only new input node of $M'[S'_2]$ can be n_1 which by construction has the same type as is the I/O type of $M[S_1]$ which by assumption is equal to the I/O type of S_2 . The argument is similar for output nodes.
- (2) **Preservation of the AND property and the OR property:** We first consider the AND property, and specifically the restriction on input edges for places. For nodes $n' \in S_2 \setminus S_1$ it is easy to see that if there are two incoming edges for n' that violate the AND property in $M'[S'_2]$ and at least one of them starts from a node in $S_2 \setminus S_1$, then the corresponding edges in M arriving in n' also violate the AND property for $M[S_2]$. For nodes $n' \in S_2 \setminus S_1$ that have two incoming edges for n' that violate the AND property for $M'[S'_2]$ such that neither of them starts from a node in $S_2 \setminus S_1$, then the corresponding edges arriving in n' also violate the AND property for $M[S_2]$. First note that for the AND property to be violated one of the edges has to start from n_1 and another from a node outside of $S_1 \cup S_2$. Then in $M[S_2]$ the n' would also be an input node and the edge from n_1 in M' would have a corresponding edge in m from a node in S_1 . If this edge is from $S_1 \cap S_2$ the AND property would be violated in $M[S_2]$ and if it is from $S_1 \setminus S_2$ then the interface nodes of $M[S_1]$ and $M[S_2]$ could not be the same. This contradiction proves that the AND property is preserved for the input nodes.

The argument for outgoing edges is analogous, and also for the incoming and outgoing edges violating the OR property.

- (3) **Preservation of acyclicity:** We show that if $M[S_2]$ is a pAND net or a 11tAND net, then there is no cycle in $M'[S'_2]$. The proof proceeds by contradiction, so we start with assuming that there is such a cycle. We consider only cycles that go through n_1 as any other cycle would also have to exist in $M[S_2]$ because of the way how contraction of $M[S_1]$ is defined. For a cycle going through n_1 there would be a corresponding cycle in $M[S_1 \cup S_2]$ since $M[S_1]$ and $M[S_1]$ is well-connected. If this corresponding cycle only goes through nodes in S_2 we would have a contradiction with the assumption that $M[S_2]$ is acyclic, so we can assume that it goes through at least one node in $S_1 \setminus S_2$. In this cycle there will be an edge (n_3, n_4) from $S_2 \setminus S_1$ to S_1 and an edge (n_5, n_6) from $S_1 \setminus S_2$ to S_2 such that the path between n_6 and n_3 only passes through nodes in S_2 . Moreover, the nodes n_4 and n_6 will be in $S_1 \cap S_2$, since edges between $S_1 \setminus S_2$ and $S_2 \setminus S_1$ are not possible because $M[S_1]$ and $M[S_2]$ have the same I/O type. Also note that n_4 is an input node of $M[S_1]$, and n_6 is an input node of $M[S_2]$. In M the nodes in $S_1 \cup S_2$ must be reachable from input nodes of M . Since we consider only internal contractions, there must therefore in M be at least one edge

(n_7, n_8) such that $n_7 \notin S_1 \cup S_2$ and $n_8 \in S_1 \cup S_2$. Consider the case where $n_8 \in S_2$. Because $M[S_2]$ is well-nested, there is in M also the edge (n_7, n_6) . This makes n_6 an input node of $M[S_1]$, and so because of the well-nestedness of $M[S_1]$, there is in M an edge (n_3, n_6) . But this leads to a contradiction with the acyclicity of $M[S_2]$ since there is in $M[S_2]$ a path from n_6 to n_3 . For the case where $n_8 \in S_1$ we cannot use the same argument as the path from n_4 to n_5 is not guaranteed to be in S_1 , but we can derive a contradiction as follows. By the well-nestedness of S_1 in M , there is an edge (n_7, n_4) in M and n_4 would therefore be an input node of $M[S_2]$. We then distinguish the subcases where $M[S_2]$ is a pAND net and where it is a 11tAND net. In the first subcase we get a contradiction with the AND property of $M[S_2]$ since n_4 is an input node with an incoming edge (n_3, n_4) . In the other subcase, where $M[S_2]$ is a 11tAND net, it holds that $n_4 = n_6$, since both are input nodes of $M[S_2]$, which has the one-input property. Consequently, the edge (n_3, n_4) and the path in $M[S_2]$ from n_6 to n_3 form a cycle in $M[S_2]$, which contradicts the acyclicity of $M[S_2]$.

- (4) **Preservation of the one-input-one-output property:** We show that if $M[S_2]$ has the one-input property, then $M'[S'_2]$ also has it. We show this by contradiction, and so we start with assuming that $M'[S'_2]$ has at least two input nodes, say n_3 and n_4 , and then show that $M[S_2]$ also has at least two input nodes. Clearly this holds if n_3 and n_4 were already input nodes of $M[S_2]$. Since only n_1 can become a new input node after contracting S_1 , the only other case to consider is where one of the nodes, say n_3 , is equal to n_1 and the other node, n_4 , is an old input node. Note that $n_4 \in S_2 \setminus S_1$, and that both n_1 and n_4 will have an incoming edge from some node not in S'_2 . As n_1 is obtained by contracting $M[S_1]$, it follows that in M the node n_4 and all the input nodes of $M[S_1]$ will have an incoming edge from a node not in $S_1 \cup S_2$. If at least one input node of $M[S_1]$ is in S_2 , then this is also an input node of $M[S_2]$. This input node is distinct from n_4 since n_4 is not in S_1 , and so $M[S_2]$ will have at least two input nodes. If, on the other hand, there are no input nodes of $M[S_1]$ in S_2 , then there must in $M[S_1]$ be a path from an input node in $S_1 \setminus S_2$ to a node in $S_1 \cap S_2$, because $S_1 \cap S_2$ is non-empty. It follows that on that path there is an edge from a node in $S_1 \setminus S_2$ to a node in $S_1 \cap S_2$. The end node of that edge must be an input node of $M[S_2]$ and is distinct from n_4 because n_4 is not in S_1 , so $M[S_2]$ has at least two input nodes.

In a similar fashion it can be shown that $M'[S'_2]$ cannot have more than one output node.

From the preservation of the type of the input and output nodes, it follows that $M'[S'_2]$ is I/O consistent, and since it is a subnet of a WF net it follows that it is itself a WF net. Since all the other properties that must hold for a WF net of a basic AND-OR type were also shown to be preserved, it follows that $M'[S'_2]$ is a WF net of basic AND-OR type. Since $S_2 \setminus S_1$ is non-empty by assumption, it follows that $M'[S'_2] = M'[(S_2 \setminus S_1) \cup \{n_1\}]$ contains at least two nodes, and so is non-trivial, and therefore contractible. \square

Lemma 21. (Case (D): preservation of contractibility)

Let M be a WF net with two contractible subnets S_1 and S_2 such that $S_1 \subseteq S_2$. Then, if M' is the result of contracting S_1 into n_1 then in M' the subnet $(S_2 \setminus S_1) \cup \{n_1\}$ is a contractible subnet.

Proof:

The proof of this lemma is identical to the proof of Lemma 20 for case (C) except where the assumption that the I/O types of $M[S_1]$ and $M[S_2]$ are identical is used to argue that there are in M no edges from $S_1 \setminus S_2$ to $S_2 \setminus S_1$, which in this proof is derived from the fact that $S_1 \subseteq S_2$. \square

In the preceding lemmas it was shown for the cases (A), (B), (C) and (D) from Figure 6 that for internal contractions the contractibility of the non-contracted net is preserved. We now combine these results to show that we have local confluence for every two internal contractions.

Theorem 22. (Local confluence up to isomorphism of the internal contraction relation)

Given a WF net M and two contractions $M \rightsquigarrow_{\text{int}} M_1$ and $M \rightsquigarrow_{\text{int}} M_2$, there are isomorphic WF nets M_3 and M_4 such that $M_1 \rightsquigarrow_{\text{int}}^* M_3$ and $M_2 \rightsquigarrow_{\text{int}}^* M_4$.

Proof:

Let $M[S_1]$ and $M[S_2]$ be the subnets of M that were contracted for $M \rightsquigarrow_{\text{int}} M_1$ and $M \rightsquigarrow_{\text{int}} M_2$, respectively. We consider all the possible cases as were defined in the beginning of this section for $M[S_1]$ and $M[S_2]$:

- (A) *They do not share nodes, but there are edges from nodes in one subnet to nodes in another.* By Lemma 18 it holds that after contracting S_1 into n_1 resulting in M_1 the subnet S_2 is contractible in M_1 . By symmetry the same holds for the subnet S_1 if we contract S_2 into n_2 resulting in M_2 . It is also clear that in both case we get identical results where all nodes from S_1 have been contracted into n_1 and all nodes from S_2 have been contracted into n_2 .
- (B) *The subnets share nodes, the corresponding WF nets have different I/O types, and each subnet has nodes that are not contained in the other.* By Lemma 19 it holds that after contracting S_1 into n_1 resulting in M_1 the subnet $S_2 \setminus S_1$ is either trivial or contractible in M_1 . By symmetry the same holds for the subnet $S_1 \setminus S_2$ in M_2 if we contract S_2 into n_2 resulting in M_2 . Moreover, it can be shown that the result of contracting first S_1 into n_1 and then, if it is non-trivial, $S_2 \setminus S_1$ into n_2 , is isomorphic to the result of first contracting S_2 into n_2 and then, if it is non-trivial, $S_1 \setminus S_2$ into n_1 . This is proven as follows.

We define the WF net M'_3 as equal to M_3 except when $S_2 \setminus S_1$ is trivial, in which case the node in $S_2 \setminus S_1$ is replaced in M_3 with n_2 . Similarly we define the WF net M'_4 as equal to M_4 except when $S_1 \setminus S_2$ is trivial, in which case the node in $S_1 \setminus S_2$ is replaced in M_4 with n_1 . It is easy to see that $M_3 \sim M'_3$, and $M_4 \sim M'_4$. We will then show that $M'_3 = M'_4$.

It is easy to see that M'_3 and M'_4 have the same nodes, namely the nodes of M minus the nodes in $S_1 \cup S_2$ plus the nodes n_1 and n_2 . Moreover, the types of n_1 and n_2 are the same in M'_3 and M'_4 , namely the I/O types in M of S_1 and S_2 , respectively. It is also clear that the sets of input and output nodes are the same as in M , since we only consider internal contractions. So it remains to be shown that the edges incident to n_1 and n_2 are the same.

We first show that the edges between n_1 and n_2 are the same in M'_3 and M'_4 . Assume there is an edge (n_1, n_2) in M'_3 . In that case there is an edge (n'_1, n'_2) in M with $n'_1 \in S_1$ and $n'_2 \in S_2 \setminus S_1$. Assume there are no output nodes of $M[S_1]$ in $S_1 \setminus S_2$, then all of them must be in $S_1 \cap S_2$. Since $S_1 \setminus S_2$ is non-empty and $M[S_1]$ is well-connected, there must be in M an edge from a node in $S_1 \setminus S_2$ to $S_1 \cap S_2$, and so there is also the edge (n_1, n_2) in M'_4 . If, on the other hand, we assume that there is at least one output node of $M[S_1]$ in $S_1 \setminus S_2$, then because of the well-nestedness of S_1 in M it follows that there is an edge from that output node to n'_2 , and so there is also in this case the edge (n_1, n_2) in M'_4 . A similar argument can be used to show that an edge (n_1, n_2) in M'_4 implies this same edge in M'_3 . Moreover, the same can be shown with an analogous argument for the edge (n_2, n_1) .

What remains to be shown is that the remaining edges incident to n_1 and to n_2 are the same. Note that there cannot be edges in M between a node in $S_1 \cap S_2$ and a node not in $S_1 \cup S_2$, since otherwise there would be a node in $S_1 \cap S_2$ that is an input node or output node of both $M[S_1]$ and $M[S_2]$, which is not possible since they have different I/O types. It follows that edges in M between nodes not in $S_1 \cup S_2$ and nodes in $S_1 \cup S_2$, are in fact either incident to a node in $S_1 \setminus S_2$ or $S_2 \setminus S_1$. So if an edge exists in M'_3 between a node not in $\{n_1, n_2\}$ and n_1 or n_2 , then that edge will exist also in M'_4 , and vice versa.

- (C) *The subnets share nodes, the corresponding WF nets have the same I/O type, and each subnet has nodes that are not contained in the other.* By Lemma 20 it holds that after contracting S_1 into n_1 resulting in M_1 the subnet $(S_2 \setminus S_1) \cup \{n_1\}$ is contractible in M_1 . By symmetry the same holds for the subnet $(S_1 \setminus S_2) \cup \{n_2\}$ in M_2 if we contract S_2 into n_2 resulting in M_2 . Moreover, it also holds that the result of contracting first S_1 into n_1 and then $(S_2 \setminus S_1) \cup \{n_1\}$ into n_3 is the same as when we first contract S_2 into n_2 and then $(S_1 \setminus S_2) \cup \{n_2\}$ into n_3 . This follows from the observation that in both cases all nodes in $S_1 \cup S_2$ are contracted into n_3 .
- (D) *One subnet is contained in the other.* Without loss of generality we assume that $S_2 \subseteq S_1$. By Lemma 21 it holds that after contracting S_2 into n_2 resulting in M_2 the subnet $(S_1 \setminus S_2) \cup \{n_2\}$ is contractible in M_2 . Moreover, it also holds that the result of contracting S_1 into n_1 is the same as when we first contract S_2 into n_2 and then $(S_1 \setminus S_2) \cup \{n_2\}$ into n_1 . This follows from the fact that in both cases all nodes in $S_1 \cup S_2$ are contracted into n_3 .

□

This brings us to the proof of Theorem 17, which states that the contraction relation has the local confluence property. We recall its content here before giving its proof.

Theorem 17. (Local confluence of the contraction relation)

For all WF nets M , M_1 and M_2 it holds that if $M \rightsquigarrow M_1$ and $M \rightsquigarrow M_2$, then there are two isomorphic a WF nets M_3 and M_4 such that $M_1 \rightsquigarrow^* M_3$ and $M_2 \rightsquigarrow^* M_4$.

Proof:

This follows from the local confluence up to isomorphism of the internal contraction relation, as stated in the preceding Theorem 22, and the observation that internal contractions on place completions and transition completions of WF nets can simulate general contractions and vice versa. More precisely, if M_1 and M_2 are tWF nets and M'_1 and M'_2 their place completions, it holds that $M_1 \rightsquigarrow M_2$ iff $M'_1 \rightsquigarrow_{\text{int}} M'_2$, and the same holds if M_1 and M_2 are pWF nets and M'_1 and M'_2 their transition completions. This observation follows straightforwardly from the definitions of (internal) contractions and completions and how they treat incoming (outgoing) edges from (to) the added nodes in the completion the same as being input (output) nodes in the original. □

6. A polynomial-time reduction algorithm

In this section we present a concrete reduction algorithm for computing the reduction of a WF net. When the reduction is computed in a naive way, where we simply iterate over all subnets to see if one of them is contractible, this could require exponential time in the size of the input workflow net. We demonstrate

that this can be done in a more sophisticated manner that leads to a polynomial-time algorithm. In addition we show how this algorithm can be used to verify if a certain WF net is an AND-OR net.

The algorithm is based on the result of Theorem 16 which states that the result of this reduction is unique up to isomorphism, no matter how we select the subnets to contract. This allows the algorithm to proceed without backtracking and essentially just repeat a process where it continues to look for contractible non-trivial subnets, and contract them, until no more such subnets can be found. The efficiency of this algorithm is improved further by an algorithm for identifying such subnets within polynomial time. As a result the whole reduction algorithm runs within polynomial time.

Algorithm 1: reduce(M)

Input: a net WF net $M = (P, T, F, I, O)$

Output: an reduction of M

```

1 foreach  $(n_1, n_2) \in (P \times P) \cup (T \times T)$  do
2   if  $n_1 = n_2$  and  $n_1 \in P$  and there exists  $t \in T$  s.t.  $t \bullet_M = \bullet_M t = \{n_1\}$  then
3     return reduce(contractSubnet( $M, \{n_1, t\}$ ));
4   if  $n_1 \neq n_2$  and  $\bullet_M n_1 = \bullet_M n_2$  and  $n_1 \bullet_M = n_2 \bullet_M$  then
5     return reduce(contractSubnet( $M, \{n_1, n_2\}$ ));
6   if  $n_1 \neq n_2$  and  $n_2$  is reachable from  $n_1$  then
7      $N \leftarrow$  expand( $M, n_1, n_2$ );
8     if  $N \neq null$  then
9       return reduce(contractSubnet( $M, N$ ));
10 return  $M$ ;

```

Non-trivial subnets, i.e., subnets with more than one node, for contraction can be found in the following way. We iterate in line 1 of Algorithm 1 over all the pairs of nodes (n_1, n_2) of the same type, i.e., pairs of places or pairs of transitions. We distinguish three cases. We start in line 2 with contracting loops, where by a loop we mean a place with a transition attached to it such that this transition has exactly two edges, one incoming from the place and one outgoing to the place. Then in line 4 we contract pairs of distinct nodes if these share the same incoming and outgoing edges. Finally, in line 6, for the remaining pairs of nodes we treat one node as an input and the other as an output. We expand forward from the input node and backward from the output node. While expanding we remember that we can discover new interface nodes. If we end up with a subnet that is of a basic AND-OR class then we contract it.

It can be shown that if any non-trivial subnet can be contracted, then at least one of our three cases will also find a subnet. This follows from the observation that for a contractible subnet it holds that either (1) it has an input node and an output node that is distinct from the input node but can be reached from it or (2) all input nodes are also output nodes and vice versa. Case (1) is covered by the test on line 6 where we assume that n_1 is an input node and n_2 an output node of the subnet we are looking for. For case (2) we can distinguish the sub-cases (a) there are two or more input/output nodes or (b) there is exactly one input/output node. In case (a) the test on line 4 will apply. In case (b) it can be observed that the nested net must be an 11pOR net if it is non-trivial. This is because it then contains a cycle, and so cannot be an 11tAND or pAND net, and it also cannot be a tOR net since the input/output node has incoming

and outgoing edges. If it is an 11pOR net then it either satisfies the test on line 2, or the net minus the input/output place defines a well-nested tOR net containing more than one node and which is covered by either the tests on line 4 or 6. So we can conclude that if there is a contractible subnet then there is a pair (n_1, n_2) that satisfies at least one of the tests on line 2, 4, or 6.

The loop runs in quadratic time. Each time we do a contraction the number of nodes decreases, so it suffices to show that we can do the expansion in polynomial time. Observe first that we can easily check in polynomial time if the output node is reachable from an input node or even enumerate the nodes in such a way that for each input node we iterate only over the output nodes reachable from it.

The expansion can be done as in Algorithm 2. Given an input and output node pair we traverse the net as in a breadth or depth-first graph search algorithm. The initial nodes for the traversal are the input and output nodes provided as parameters, yet we make sure that we traverse forward only if the currently inspected node is not an output node (see lines 7 through 12) and that we traverse backward only if the currently inspected node is not an input node (see lines 13 through 18). The newly encountered nodes are tested to check if they can be input (output) nodes of the subnet we are looking for. This is done by testing if (1) they have incoming (outgoing) edges from (to) exactly the same nodes as the initial input (output) node and (2) are input node of M iff the initial input node is an input node of M . When this test succeeds, the node is added to the input (output) set of the constructed net (see lines 9 and 15).

It is easy to see that during the traversal we discover only nodes that must be in any well-nested subnet with the provided interface nodes. This holds because, if there is in M an edge (n_1, n_2) , and n_1 is in a subnet that is a WF net, but cannot be an output node of that WF net, then n_2 is also in that subnet. So we argue that the algorithm finds only non-trivial subnets that are well-nested and of a basic AND-OR class. Otherwise it returns *null*.

First, observe that in lines 7 and 13 we check if a node has incoming (outgoing) edges from (to) exactly the same nodes as the initial input (output) node. However, for a well-nested subnet S it is required that all nodes in I_S (O_S) must have incoming (outgoing) edges from (to) the same nodes *outside* S . We argue that our check is correct, which can be reasoned by examining all the basic AND-OR classes. For pAND and 11tAND nets no cycles are allowed, so for input (output) nodes all incoming (outgoing) edges must come from outside. In tOR nets transitions have at most one incoming (outgoing) edges internally and none if they are input (output) node, so all incoming edges come from outside. Finally, for 11pOR nets only the initial single input and single output nodes are possible and no other internal node can have edges from outside. Now we know that if a subnet is found, it must be a WF net, since (1) M is assumed to be a WF net, (2) all the input (output) nodes are guaranteed to be connected to the same nodes outside and either all input (output) node of M or none, and (3) we include all the nodes reachable forwards (backwards) from any of the input (output) nodes.

Next, during traversal, we maintain the set of possible basic AND-OR classes that are still compatible with the discovered net. It is initiated in the lines 3 through 4, can be updated in lines 8 and 14 if the expanded net is observed to not to be one input one output net, respectively, and in lines 21 and 22 if the AND or OR properties, respectively, stop to be satisfied. Note that if n' here satisfies the AND and OR properties it will continue to do so, since all its predecessors and successors have already been added to S in the preceding steps on lines 12 and 18 unless it is an input or output node. Finally, on line 23 we examine the acyclicity property, which is required for AND nets. This guarantees that we know if the subnet is of a basic AND-OR class. If no compatible basic AND-OR classes are left (see line 24), a *null* is returned.

At the end of the algorithm, on line 25 the found subnet is returned, which is spanned by S and

Algorithm 2: $\text{expand}(M, i, o)$ **Input:** a WF net $M = (P, T, F, I, O)$ and nodes $i, o \in P$ or $i, o \in T$ **Output:** a subnet S of M such that $M[S]$ is a well-nested WF net of a basic AND-OR class with at least one input node i and at least one output node o , or *null* if such a subnet does not exist

```

1  $(S, I_S, O_S) \leftarrow (\{i, o\}, \{i\}, \{o\});$  // initialize subnet  $S$  and input/output sets of  $M[S]$ 
2  $TBA \leftarrow \{i, o\};$  // nodes to be analysed
3 if  $i, o \in P$  then  $PC \leftarrow \{11pOR, pAND\};$  // init. set of possible
4 if  $i, o \in T$  then  $PC \leftarrow \{11tAND, tOR\};$  // basic AND-OR classes
5 while  $TBA \neq \emptyset$  and  $PC \neq \emptyset$  do
6    $n' \leftarrow$  pick and remove from  $TBA$ ;
7   if  $\bullet_M n' = \bullet_M i$  and  $n' \in I \Leftrightarrow i \in I$  then // is  $n'$  input node?
8     if  $n' \notin I_S$  then  $PC \leftarrow PC \setminus \{11pOR, 11tAND\};$  // new input
9      $I_S \leftarrow I_S \cup \{n'\};$ 
10  else
11     $TBA \leftarrow TBA \cup (\bullet_M n' \setminus S);$  // add predecessors
12     $S \leftarrow S \cup \bullet_M n'$ ;
13  if  $n' \bullet_M = o \bullet_M$  and  $n' \in O \Leftrightarrow o \in O$  then // is  $n'$  output node?
14    if  $n' \notin O_S$  then  $PC \leftarrow PC \setminus \{11pOR, 11tAND\};$  // new output
15     $O_S \leftarrow O_S \cup \{n'\};$ 
16  else
17     $TBA \leftarrow TBA \cup (n' \bullet_M \setminus S);$  // add followers
18     $S \leftarrow S \cup n' \bullet_M$ ;
19  if  $(n' \notin I_S \wedge |\bullet_{M[S]} n'| \neq 1)$  or  $(n' \in I_S \wedge |\bullet_{M[S]} n'| \neq 0)$  or
20   $(n' \notin O_S \wedge |n' \bullet_{M[S]}| \neq 1)$  or  $(n' \in O_S \wedge |n' \bullet_{M[S]}| \neq 0)$  then
21    if  $n' \in P$  then  $PC \leftarrow PC \setminus \{pAND, 11tAND\};$  // not AND net
22    if  $n' \in T$  then  $PC \leftarrow PC \setminus \{tOR, 11pOR\};$  // not OR net
23 if  $M[S]$  cyclic then  $PC \leftarrow PC \setminus \{pAND, 11tAND\};$  // AND is acyclic
24 if  $PC = \emptyset$  then return null; // no more possible basic classes?
25 return  $S$ ;

```

is a well-nested, internal subnet of M that is in at least one basic AND-OR class. This concludes the argument for correctness of Algorithm 2.

From the correctness it follows that it can be used to check if a WF net is an AND-OR net by checking if it is reduced to a trivial WF net. After all, if this happens the reduction of the WF net can be executed in reverse and defines a way of generating the WF net by substitutions of basic AND-OR nets. Indeed, it can be shown that for every AND-OR net there is a generation possible that uses only basic AND-OR nets, and therefore it is reduced to a trivial net by AND-OR reduction. The argument is as follows. Let us denote the generation of an AND-OR net as a nested expression consisting of the substitution expressions $M \otimes_n N$ combined with basic AND-OR nets. As noted earlier, this operation is essentially associative in the sense that $A \otimes_{n_1} (B \otimes_{n_2} C)$ and $(A \otimes_{n_1} B) \otimes_{n_2} C$ produce the same result, if n_1 appears in A and n_2 appears in B . As a consequence, any expression that describes the generation of an AND-OR net can be rewritten to an equivalent expression of the form $(\dots ((N_0 \otimes_{n_1} N_1) \otimes_{n_2} N_3) \dots \otimes_{n_k} N_k)$ where N_0 is a trivial net containing the node n_1 and all other N_i are basic AND-OR nets. When read from left to right, this expression will represent a generation of the AND-OR net while substituting only basic AND-OR

nets.

7. Summary and Future Research

In this paper we introduce a notion of reduction, which reduces a WF net to a smaller net by iteratively contracting certain well-formed subnets into single nodes until no more such contractions are possible. This reduction is interesting for several reasons. The first reason is that it preserves the soundness and unsoundness of the net, so can be used to help users understand why a WF net is problematic. It might also give some valuable insights for determining the different possible decompositions, e.g., how parts of the workflow can be best distributed to independent organisational units or, in case of workflows representing computations, e.g., as in scientific workflows, to different servers. The second reason is that it can provide as a side-effect a hierarchical structure of parts, or the whole, of the WF net, which can help user to understand the structure or large WF nets. The third and final reason is that the reduction can be used to show that a certain WF net is an AND-OR net, because in that case the net is reduced to a one-node WF net. This class of WF nets was introduced in earlier work, and arguably describes nets that follow good hierarchical design principles which can be compared to structured design of programs and using well nested procedures and functions, rather than unrestricted goto statements. As was shown in earlier work, these nets have the desired soundness property.

It is shown that the reduction is confluent up to isomorphism, which means that despite the inherent non-determinism that comes from the choice of subnets that are contracted, the final result of the reduction is always the same up to the choice of the identity of the nodes. Based on this result, an algorithm is presented that computes the reduction, and runs in polynomial time.

As a byproduct of the reduction procedure, a refinement tree for the hierarchical structure of the net can be constructed, like has been done for similar classes of nets [2, 4, 3]. It is worth to investigate how such refinement trees can be used to determine efficiently sound markings and to model recovery regions. Moreover, it can be investigated to what extent this hierarchy is unique, or could be made so by normalising it, since that could make it more effective as a tool for understanding the structure of a net. For example, in a refinement tree, if a parent and a child both represent contractions of AND nets, then they can be merged into a single contraction of a larger AND net. Another source of ambiguity comes from the observation that linear nets are simultaneously AND and OR nets. It would be interesting to investigate if these types of ambiguity capture all ambiguity.

Finally, another direction for future research is extending the class of sound free-choice nets that can be generated as AND-OR nets by having additional substitution rules for edges. We could define ptAND, tpAND, ptOR and tpOR nets where the small letters indicate the type of input and output nodes, respectively. So an edge from a place to a transition could be replaced with a tpAND or tpOR net. Note that the original place and transition remains present. If the ptAND is a one-output net, and the ptOR net is a one-input net, this preserves substitution soundness, and also the free-choice property. It would be interesting to investigate to what extent this would come close to generating all choice-free substitution-sound WF nets.

Acknowledgements

This research was sponsored by National Science Centre based on decision DEC-2012/07/D/ST6/02492.

References

- [1] E. Best, R. R. Devillers, and J. Esparza. General refinement and recursion operators for the Petri box calculus. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *STACS 93, Proc. of the 10th Annual Symposium on Theoretical Aspects of Computer Science*, volume 665 of *Lecture Notes in Computer Science*, pages 130–140. Springer, February 25-27 1993.
- [2] P. Chrzastowski-Wachtel, B. Benatallah, R. Hamadi, M. O’Dell, and A. Susanto. A top-down Petri net-based approach for dynamic workflow modeling. In *Proceedings of BPM’03*, pages 336–353. Springer-Verlag, 2003.
- [3] P. Chrzastowski-Wachtel, P. Golab, and B. Lewinski. Sound recoveries of structural workflows with synchronization. In M. S. Szczuka, L. Czaja, and M. Kacprzak, editors, *Proceedings of the 22nd International Workshop on Concurrency, Specification and Programming, Warsaw, Poland*, volume 1032 of *CEUR Workshop Proceedings*, pages 73–87. CEUR-WS.org, 2013.
- [4] P. Chrzastowski-Wachtel. Determining sound markings in structured nets. *Fundam. Inf.*, 72(1-3):65–79, Apr. 2006.
- [5] D. De Roure, C. Goble, and R. Stevens. The design and realisation of the virtual research environment for social sharing of workflows. *Future Generation Computer Systems*, 25(5):561 – 567, 2009.
- [6] J. Dehnert and P. Rittgen. Relaxed soundness of business processes. In K. Dittrich, A. Geppert, and M. Norrie, editors, *Advanced Information Systems Engineering*, volume 2068 of *Lecture Notes in Computer Science*, pages 157–170. Springer Berlin Heidelberg, 2001.
- [7] R. Devillers, H. Klaudel, and R. Riemann. General refinement for high level Petri nets. *Foundations of Software Technology and Theoretical Computer Science*, pages 297–311, 1997.
- [8] J. Esparza and M. Silva. Top-down synthesis of live and bounded free choice nets. In G. Rozenberg, editor, *Advances in Petri Nets 1991, Papers from the 11th International Conference on Applications and Theory of Petri Nets*, volume 524 of *Lecture Notes in Computer Science*, pages 118–139. Springer, 1990.
- [9] D. Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. Analysis on demand: Instantaneous soundness checking of industrial business process models. *Data Knowl. Eng.*, 70(5):448–466, May 2011.
- [10] C. Flender and T. Freytag. Visualizing in the soundness of workflow nets. In *Bericht 267, Tagungsband des 13. Workshops Algorithmen und Werkzeuge für Petri-Netze, AWPN’06, FBI-HH-B-267/06*, pages 47–53, Sep, 06.
- [11] R. Gold. Reductions of control flow graphs. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 8(3):427–434, 2014.
- [12] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821, Oct. 1980.
- [13] G. Keller, M. Nuttigens, and A.-W. Scheer. Semantische prozessmodellierung auf der grundlage: Ereignis-gesteuerter prozessketten (EPK). *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, 89, Januar 1992.
- [14] A. Martens. On compatibility of web services. *Petri Net Newsletter*, 65:12–20, 2003.
- [15] A. Martens. Analyzing web service based business processes. In *Proceedings of the 8th International Conference, Held As Part of the Joint European Conference on Theory and Practice of Software Conference on Fundamental Approaches to Software Engineering, FASE’05*, pages 19–33, Berlin, Heidelberg, 2005. Springer-Verlag.
- [16] T. Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–580, Apr 1989.

- [17] Object Management Group (OMG). Business Process Model and Notation (BPMN) Version 2.0. Technical report, <http://www.omg.org/spec/BPMN/2.0/>, January 2011.
- [18] C. A. Petri and W. Reisig. Petri net. *Scholarpedia*, 3(4):64–77, 2008.
- [19] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas. Structuring acyclic process models. *Information Systems*, 37(6):518 – 538, 2012.
- [20] A. Polyvyanyy, M. Weidlich, and M. Weske. Connectivity of workflow nets: the foundations of stepwise verification. *Acta Inf.*, 48(4):213–242, 2011.
- [21] F. Puhmann and M. Weske. Investigations on Soundness Regarding Lazy Activities. In S. Dustdar, J. Fideiro, and A. Sheth, editors, *International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 145–160. Springer-Verlag, Berlin, 2006.
- [22] J. Sroka, P. Chrzastowski-Wachtel, and J. Hidders. On generating *-sound nets with substitution. *Application of Concurrency to System Design, International Conference on*, 0:3–12, 2011.
- [23] J. Sroka and J. Hidders. On generating *-sound nets with substitution. *Inf. Syst.*, 40:32–46, 2014.
- [24] I. Suzuki and T. Murata. A method for stepwise refinement and abstraction of Petri nets. *Journal of Computer and System Sciences*, 27(1):51–76, 1983.
- [25] N. Trčka, W. M. Aalst, and N. Sidorova. Data-flow anti-patterns: Discovering data-flow errors in workflows. In *Proceedings of the 21st International Conference on Advanced Information Systems Engineering, CAiSE '09*, pages 425–439, Berlin, Heidelberg, 2009. Springer-Verlag.
- [26] W. van der Aalst, G. D. Michelis, and C. Ellis, editors. *Structural Analysis of Workflow Nets with Shared Resources*, volume volume 98/7 of *Computer science reports*, 1998.
- [27] W. M. P. van der Aalst. The application of Petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.
- [28] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, July 2003.
- [29] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Asp. Comput.*, 23(3):333–363, 2011.
- [30] R. van der Toorn. *Component-based software design with Petri nets: an approach based on inheritance of behavior*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2004. PhD thesis.
- [31] K. van Hee, N. Sidorova, and M. Voorhoeve. Generalised Soundness of Workflow Nets Is Decidable. In J. Cortadella and W. Reisig, editors, *Application and Theory of Petri Nets 2004*, volume 3099 of *Lecture Notes in Computer Science*, pages 197–215. Springer-Verlag, Berlin, 2004.
- [32] K. M. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and separability of workflow nets in the stepwise refinement approach. In W. M. P. van der Aalst and E. Best, editors, *ICATPN*, volume 2679 of *Lecture Notes in Computer Science*, pages 337–356. Springer, 2003.
- [33] J. Vanhatalo, H. Völzer, and J. Koehler. The refined process structure tree. In M. Dumas, M. Reichert, and M.-C. Shan, editors, *Business Process Management*, volume 5240 of *Lecture Notes in Computer Science*, pages 100–115. Springer Berlin Heidelberg, 2008.
- [34] J. Vanhatalo, H. Völzer, and J. Koehler. The refined process structure tree. *Data Knowl. Eng.*, 68(9):793–818, Sept. 2009.

- [35] H. M. W. Verbeek. *Verification of WF-nets*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2004. PhD thesis.
- [36] H. M. W. Verbeek, T. Basten, and W. M. P. van der Aalst. Diagnosing workflow processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
- [37] Z. Wang and D. Wei. Modeling complex system using T-subnet based hierarchical Petri nets. *Journal of Computers*, 4(9):829–836, Sep 2009.