Correctness-Preserving Configuration of Business Process Models

W.M.P. van der Aalst[1,2], M. Dumas[2], F. Gottschalk[1], A.H.M. ter Hofstede[2], M. La Rosa[2], and J. Mendling[2]

[1] Eindhoven University of Technology, The Netherlands.
{w.m.p.v.d.aalst,f.gottschalk}@tue.nl
[2] Queensland University of Technology, Australia.
{m.dumas,a.terhofstede,m.larosa,j.mendling}@qut.edu.au

Abstract. Reference process models capture recurrent business operations in a given domain such as procurement or logistics. These models are intended to be configured to fit the requirements of specific organizations or projects, leading to individualized process models that are subsequently used for domain analysis or solution design. Although the advantages of reusing reference process models compared to designing process models from scratch are widely accepted, the methods employed to configure reference process models are manual and error-prone. In particular, analysts are left with the burden of ensuring the correctness of the individualized process models and to manually fix errors. This paper proposes a foundation for configuring reference process models incrementally and in a way that ensures the correctness of the individualized process models, both with respect to syntax and behavioral semantics. Specifically, assuming the reference process model is behaviorally sound, the individualized process models are guaranteed to be sound.

Key words: Reference process model, model configuration, Petri net.

1 Introduction

The design of business process models is labor-intensive, especially when such models are required to be detailed enough to support the development of software systems. To avoid the effort of creating process models from scratch, several consortia and vendors have defined so-called *reference process models*. These models capture proven practices and recurrent business operations in a given domain. They are designed in a generic manner and are intended to be individualized to fit the requirements of specific organizations or IT projects. Commercial process modeling tools come with standardized libraries of reference process models such as the IT Infrastructure Library (ITIL) [21] or the Supply Chain Operations Reference (SCOR) model [20]. Also, the SAP Reference Model [6] incorporates a collection of process models corresponding to common business operations supported by SAP's Enterprise Resource Planning (ERP) system.

Reference process models in commercial use lack an explicit representation of configuration alternatives and decisions. As a result, their individualization is entirely manual [18]. Analysts take the reference models merely as a source of inspiration, but ultimately, they design their own model on the basis of the reference model, with little guidance as to which model elements need to be removed, added or modified to address a given requirement. To address this shortcoming, we introduced in previous work the concept of *configurable process models* [18]. A configurable process model represents multiple variants of a business process model in an integrated manner. In line with methods from the field of software product lines [17], these alternatives are captured as *variation points.* For example, the fact that a task in a reference process model may or may not appear in an individualized model is captured by attaching a variation point to that task which can take the value "On" or "Off". Individualized models are obtained from configurable models by selecting values for each variation point.

While configurable process models provide guidance to analysts during individualization, they do not guarantee that the individualized models are correct, whether syntactically or semantically. For example, if a model element or an entire path in a reference process model is removed during configuration, the remaining model elements need to be re-connected to maintain syntactic correctness. Also, the configuration of variation points attached to parallel splits, decision points and synchronization points in a configurable process model may lead to the introduction of deadlocks. And if the individualized process model contains such semantic errors, it needs to be manually fixed.

The contribution of this paper is a framework for configuring reference process models in a correctness-preserving manner. The framework includes a technique to derive propositional logic constraints that, if satisfied by a configuration step, guarantee the syntactic correctness of the resulting model. We prove that for a large class of process models, these constraints also ensure that semantic correctness is preserved. The framework supports *staged configuration* [8]. In other words, it allows correctness to be checked at each intermediate step of the configuration procedure. Whenever a value is assigned to a variation point, the current set of constraints is evaluated. If the constraints are satisfied, the configuration step is applied. If on the other hand the constraints are violated, we compute a reduced propositional logic formula, from which we can identify additional variation points that need to be configured simultaneously in order to preserve correctness (e.g. if an edge in the process model is removed, all nodes in a path starting with that edge need to be removed). The set of constraints is incrementally updated after each step of the configuration procedure.

The proposal is intended as a foundation for reference process model configuration. Accordingly, we adopt a Petri net-based representation of process models, thus abstracting from the specificities of process modeling notations used in practice (e.g. UML Activity Diagrams, EPC, BPMN). We use a class of Petri nets, namely workflow nets, which are specifically designed to represent business processes [1]. Workflow nets come with a notion of behavioral correctness known as soundness, which ensures the absence of deadlocks and improper completion. In this paper, we enhance workflow nets with the notion of variation point, leading to the concept of a configurable workflow net. We then define a notion of configuration step over such nets and we show how to derive correctness-preserving constraints for such steps. A core result of the paper is that, for workflow nets that satisfy the "free-choice" property [9], if the outcome of a configuration step starting from a sound workflow net is a workflow net, then this latter workflow net is sound. This means that for this class of nets, configuration steps that preserve syntactic correctness also preserve behavioral correctness.

The paper is structured as follows. Section 2 introduces workflow nets and the notion of soundness while Section 3 introduces the notion of configurable workflow net and configuration step. Section 4 discusses the derivation of constraints that guarantee the preservation of syntactic correctness, and proves that these constraints also guarantee soundness for free-choice nets. The paper concludes with a section on related work, a summary, and an outlook on open issues.

2 Background

Petri nets are a formal model of concurrent systems [16]. Petri nets benefit from a rich body of theoretical results, analysis techniques and tools. They have been extensively applied to the formal verification of business process models [23]. These features make Petri nets suitable for establishing a formal foundation for business process model configuration. In addition, mappings exist between process modeling languages used in practice (e.g. UML Activity Diagrams, EPC, BPMN, BPEL) and Petri nets. These mappings provide a basis for extending the results outlined in this paper to concrete process modeling notations.

We use a class of Petri nets, namely workflow nets, specifically designed for business process modeling. Workflow nets have a single starting point and ending point, which captures the intuition that business processes are instantiated, and each process instance progresses independently through a series of activities until completion. A desirable property is that an instance of a workflow net always completes properly. This is captured by the notion of soundness. To make the paper self-contained, we provide an introduction to workflow nets and soundness.

2.1 Workflow nets: Syntax

Petri nets are composed of two types of elements, namely transitions and places, connected by directed arcs. Transitions represent tasks while places represent the status of the system before or after the execution of a transition. Formally:

Definition 1 (Petri net, Preset, Postset). *A Petri net is a triple $PN = (P, T, F)$, such that:*

- P *is a finite set of places,*
- T *is a finite set of transitions* ($P \cap T = \varnothing$),
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation).*

For each node $x \in P \cup T$, we use $\bullet x$ and $x \bullet$ to denote the set of inputs to x (preset) and the set of outputs of x (postset). □

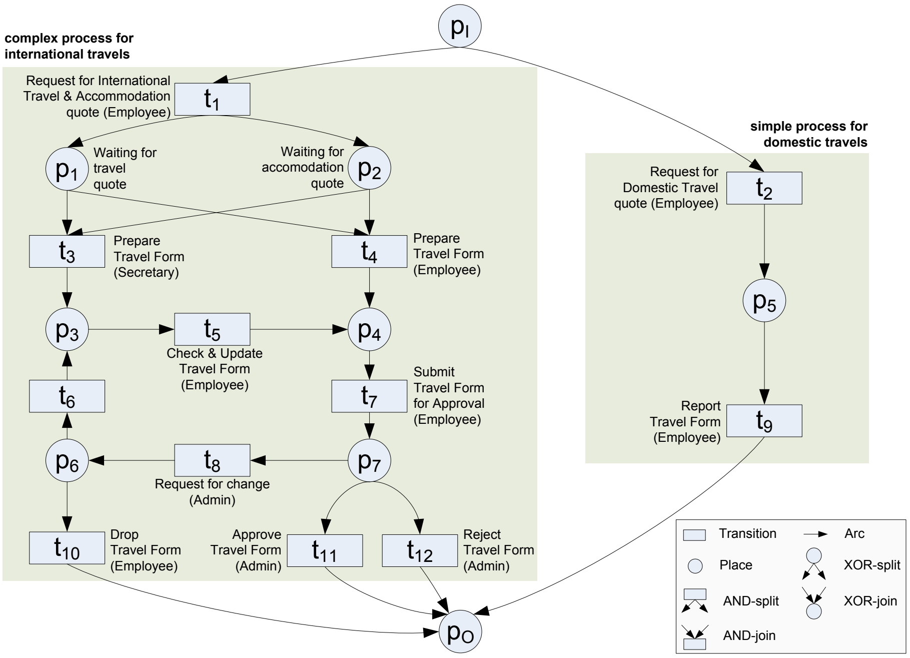
p_I
complex process for international travels
Request for International Travel & Accommodation quote (Employee)
t1
Waiting for travel quote
p1
Waiting for accomodation quote
p2
t3
Prepare Travel Form (Secretary)
t4
Prepare Travel Form (Employee)
p3
t5
p4
Check & Update Travel Form (Employee)
t6
t7
Submit Travel Form for Approval (Employee)
p6
t8
p7
Request for change (Admin)
t10
Drop Travel Form (Employee)
Approve Travel Form (Admin)
t11
t12
Reject Travel Form (Admin)
p_O
simple process for domestic travels
Request for Domestic Travel quote (Employee)
t2
p5
Report Travel Form (Employee)
t9
Transition
Arc
Place
XOR-split
AND-split
XOR-join
AND-join

Fig. 1. Reference model for travel form approval.

Fig. 1 shows a process model for travel requisition approval as a Petri net. It consists of two variants: the left one for international travel and the right one for domestic travel. After requesting a quote for international travel, either the employee or an assistant prepares the travel requisition form. In case of the latter, the employee needs to check the form before submitting it for approval. The administrator can then approve or reject the requisition, or make a request for change. At this point, the employee can update the form according to the administrator's suggestions and re-submit it, or drop the case. In contrast, the application for domestic travel only requires the employee to ask for a quote and to report the travel requisition to the administration.

A business process model may be executed a number of times to deal with different cases (e.g. different travel requests in the example). Each of these cases (called *process instances*) has a distinct start (input) and an end (output). Accordingly, we are only interested in Petri nets with a unique source place (representing the input) and a unique sink place (output), and such that all other nodes are on a directed path between the input and the output places. A Petri net satisfying these conditions represents a *structurally correct* process model and is known as a *workflow net* [1]. Formally:

Definition 2 (Workflow net). *Let $PN = (P, T, F)$ be a Petri net and F^* is the reflexive transitive closure of F. PN is a workflow net (WF-net) iff:*

- *there exists exactly one $p_I \in P$ such that $\bullet p_I = \varnothing$, and*

- *there exists exactly one* $p_O \in P$ *such that* $p_O \bullet = \varnothing$, *and*
- *for all* $n \in P \cup T$, $(p_I, n) \in F^*$ *and* $(n, p_O) \in F^*$. □

The Petri net in Fig. 1 is a *WF*-net.

2.2 Workflow nets: Semantics

Behavioral correctness of a *WF*-net is defined with respect to the states that a process instance can be in during its execution. A state of a *WF*-net is represented by the marking of its places with tokens. In other words, in a given state, each place is either empty, or it contains one or more tokens (i.e. it is marked). A transition is enabled in a given marking, if all the places in the transition's preset are marked. Once enabled, the transition can fire (i.e. can be executed) by removing a token from each place in the preset and putting a token into each subsequent place of the transition's postset. This leads to a new state. Formally:

Definition 3 (Marking, Enabling Rule, Firing Rule). *Let* $N = (P, T, F)$ *be a WF-net with source place* p_I *and sink place* p_O*:*

- $M : P \rightarrow \mathbb{N}$ *is a marking of* N *and* $\mathbb{M}(N)$ *is the set of markings of* N,
- M_I *is the initial marking of* N *with one token in place* p_I, *i.e.* $M_I = [p_I]$,
- M_O *is the final marking of* N *with one token in place* p_O, *i.e.* $M_O = [p_O]$,
- $M(p)$ *returns the number of tokens in place* p *if* $p \in dom(M)$,
- *For any two markings* $M, M' \in \mathbb{M}(N)$, $M \geq M'$ *iff* $\forall_{p \in P}\ M(p) \geq M'(p)$,
- *For any transition* $t \in T$ *and any marking* $M \in \mathbb{M}(N)$, t *is* enabled *at* M, *denoted as* $M[t\rangle$, *iff* $\forall_{p \in \bullet t}\ M(p) \geq 1$. *Marking* M' *is reached from* M *by firing* t *and* $M' = M - \bullet t + \bullet t$,
- *For any two markings* $M, M' \in \mathbb{M}(N)$, M' *is reachable from* M *in* N, *denoted as* $M' \in N[M\rangle$, *iff there exists a firing sequence* $\sigma = \langle t_1, t_2, ..., t_n \rangle$ *leading from* M *to* M', *and we write* $M \overset{\sigma}{\twoheadrightarrow}_N M'$. *If* $\sigma = \langle t \rangle$, *we use the notation* $M \xrightarrow{t}_N M'$. N *can be omitted if clear from the context.* □

The execution of a process instance starts with the state in which the input place has one token and no other place is marked. The execution of this process instance should then progress through transition firings until a proper completion state. This intuition is captured by three requirements [1]. Firstly, every process instance should always have the option to complete. If a *WF*-net satisfies this requirement, it will never run into a deadlock or livelock. Secondly, every process instance should eventually reach the state in which there is one token in the output place p_O, and no tokens are left behind in any other place, since this would signal that there is still work to be done. Thirdly, for every transition, there should be at least one execution sequence from the initial marking (where only p_I is marked) to the final marking (where only p_O is marked) that includes at least one firing of this transition. In other words, no transition in the *WF*-net should be spurious. A *WF*-net fulfilling these requirements is *sound*. Formally:

Definition 4 (Sound *WF*-net). *Let* $N = (P, T, F)$ *be a WF-net and* M_I, M_O *be the initial and end markings.* N *is sound iff:*

- *option to complete: for every marking M reachable from M_I, there exists a firing sequence leading from M to M_O, i.e.* $\forall_{M \in N[M_I\rangle}\ M_O \in N[M\rangle$*, and*
- *proper completion: the marking M_O is the only marking reachable from M_I with at least one token in place p_o, i.e.* $\forall_{M \in N[M_I\rangle}\ M \geq M_O \Rightarrow M = M_O$*,*
- *no dead transitions: every transition can be reached by the initial marking, i.e.* $\forall_{t \in T}\ \exists_{M \in N[M_I\rangle}\ M[t\rangle$. □

3 Process Model Configuration

There are several ways to capture variation points for the purpose of representing a configurable process model [18, 7, 11]. In this paper we choose the approach presented in [11], which is based on the concept of inheritance of process behavior [2], since it abstracts from vendor-specific process modeling notations and can easily be applied to Petri nets. Accordingly, we define the notion of *configurable WF-net*, where each transition captures a variation point whose possible values (or *variants*) are: *allowed*, *hidden* and *blocked*.

Hiding a transition refers to skipping its execution while it is fired, without affecting the rest of the process flow. Consider for example the *WF*-net in Fig. 1. Some organizations may not require a quote for domestic travels. Thus, the task to request a quote can be skipped from the process model by hiding transition t_2. The process continues without forcing the employee to request a quote.

Blocking a transition implies to inhibit it in the process model. Blocked transitions cannot forward cases and all the subsequent transitions will never be executed if they cannot be enabled via other paths. For example, if t_2 in Fig. 1 is blocked, the process for domestic travels cannot be triggered and all travel approvals must be done via the complex variant.

If a transition is neither blocked nor hidden, we say it is allowed, meaning nothing changes in the model. To configure a *WF*-net each transition has to be assigned one value among hidden, blocked or allowed. Formally:

Definition 5 (Configuration). *Let $N = (P, T, F)$ be a WF-net, then $c_N \in T \to \{allow, hide, block\}$ is a* configuration *for N. We define:*

- $E^c_N = \{t \in T \mid c(t) = allow\} \subseteq T$ *as the set of all allowed transitions,*
- $H^c_N = \{t \in T \mid c(t) = hide\} \subseteq T$ *as the set of all hidden transitions,*
- $B^c_N = \{t \in T \mid c(t) = block\} \subseteq T$ *as the set of all blocked transitions.*[1]

If N is clear from the context, we drop the subscript. □

Based on these configuration values, a configured net is obtained representing the new behavior of the process model. This new Petri net is a restriction of the behavior of the starting model (the reference model), where all the hidden transitions are replaced by silent τ transitions and all the blocked transitions are removed. Also, all the places connected only to blocked transitions and all the flow relations from/to blocked transitions have to be removed too. Formally:

[1] $E^c_N \cap H^c_N \cap B^c_N = \varnothing$ follows from the definition of N.

Definition 6 (Configured net). *Let $N = (P, T, F)$ be a WF-net and let c be a configuration of N. The resulting configured net $N^c(P^c, T^c, F^c)$is defined as follows:*

- $T^c = (T \setminus (B^c \cup H^c)) \ \cup \ \{\tau_t \mid t \in H^c\}$,
- $F^c = (F \ \cap \ ((P \cup T^c) \times (P \cup T^c))) \ \cup \ \{(p, \tau_t) \mid (p,t) \in F \wedge t \in H^c\} \ \cup \ \{(\tau_t, p) \mid (t,p) \in F \wedge t \in H^c\}$,
- $P^c = (P \cap \bigcup_{(x,y) \in F^c} \{x, y\}) \cup \{p_I, p_O\}$. □

As an example, Fig. 2a shows a configuration derived from the *WF*-net in Fig. 1, where the transitions t_2 and t_9 have been blocked to allow the complex approval process only. In this configuration employees have to prepare the approval form on their own, as t_3 has been blocked, and cannot drop a form application if a change is requested after approval (t_{10} also blocked). Place p_5 has been removed as it became disconnected after removing t_2 and t_9.

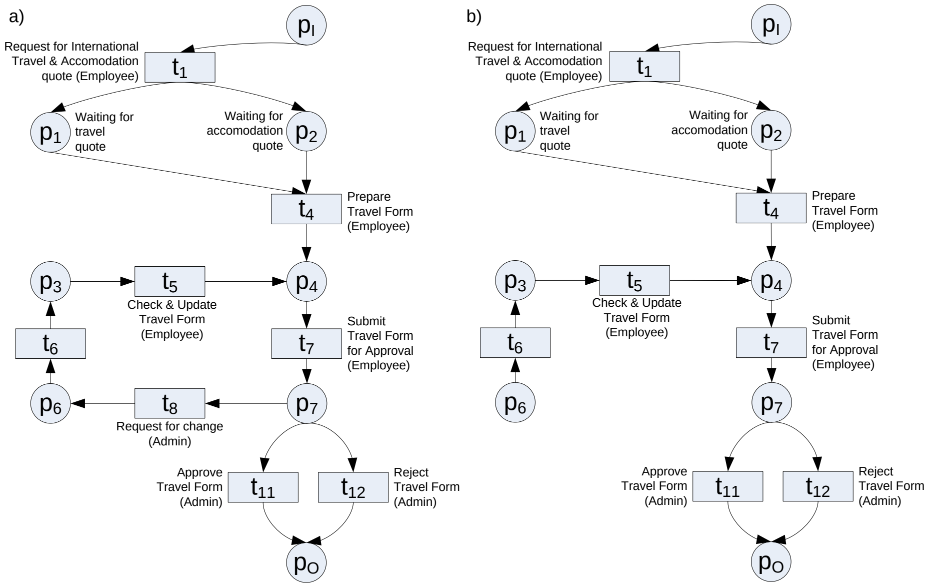
a)
b)
p_I
Request for International Travel & Accomodation quote (Employee)
t_1
Waiting for travel quote
p_1
Waiting for accomodation quote
p_2
t_4
Prepare Travel Form (Employee)
p_3
t_5
Check & Update Travel Form (Employee)
p_4
t_6
t_7
Submit Travel Form for Approval (Employee)
p_6
t_8
Request for change (Admin)
p_7
Approve Travel Form (Admin)
t_11
t_12
Reject Travel Form (Admin)
p_O

Fig. 2. a) Correct process configuration b) Incorrect process configuration.

A process configuration has to comply with the requirements of the domain. This may prevent users from configuring the values of transitions freely. For example, in the travel management domain, if an employee submits a travel form for approval there must be at least an option to accept the request and an option to reject it. This is clearly a requirement of the domain, which forbids users to block both t_{11} and t_{12} in the process model. In [14] we showed how propositional logic expressions can be used to encode domain constraints. By evaluating each transition's value against these constraints with a SAT solver, it is possible to prevent all the configurations which would violate the constraints.

Nonetheless, the set of constraints derived from the domain are in most cases not sufficient to guarantee the syntactic and semantic correctness of the configured model. Indeed, as per Definition 6, a configured net can be any Petri net, which means that it can contain elements that are not on a path from p_I to p_O, or which are completely disconnected. For example, forbidding the request for a change by blocking t_9 in the *WF*-net of Fig. 2a would make p_6, t_6, p_3 and t_5 unreachable, yielding the net of Fig. 2b. Such a configuration is not syntactically correct and hence not semantically correct either, according to Definition 4. So, as soon as t_3 and t_8 are blocked, it would be desirable to suggest the user to block t_6 and t_5 too, so as to get rid of the unreachable branch. In the following section we present an approach to automatically derive a set of constraints from a *WF*-net that preserve the model correctness during its configuration.

4 Correctness-Preserving Configuration

Existing tools like Woflan [23] support the verification of Petri net-based process models. These tools could be used to check every single configured net that can be derived from a reference process model. If the net is incorrect, the configuration that has generated this net should be excluded from the set of possible configurations. However, this approach is costly, considering that reference process models can potentially yield thousands of individualized process models.

Our aim is therefore to define a framework which allows incorrect configuration steps to be discarded incrementally and without computing all possible configurations of the reference model. In addition, the framework needs to seamlessly integrate the domain constraints, so that a user can derive a correct process model which is also satisfies any domain constraints.

To this end, we complement the domain constraints with a set of process constraints to guarantee the preservation of syntactic and semantic correctness in the configured net. Both sets of constraints are captured in propositional logic over the nodes of a *WF*-net and are reduced by a BDD solver. In this way we can provide interactive support to the user, by pinpointing the impact of each configuration step on the resulting net and by eliminating unfeasible options.

4.1 Preserving syntactic correctness

In a staged configuration, users make configuration decisions one after another in steps, and the set of configuration options is recalculated after each step. To remain syntactically correct, a *WF*-net must thus be checked on which configuration options are still viable among the transitions that have not been configured yet. For this, we have to consider the configuration decisions already taken.

To distinguish nodes which remain in the net from nodes which do not, we use a boolean variable for each node. If the variable is set to *true*, the node remains part of the net; if it is set to *false*, the node is dropped in the configured net. Accordingly, we assign a blocked transition the value *false*, while a transition that is allowed or hidden is assigned the value *true*. Since silent transitions have the same routing behavior as the original transitions, we do not need to distinguish hidden from allowed transitions. All transitions that are not explicitly configured remain as variables (i.e. unset).

According to Definition 6, any internal place remains in the net if there is a non-blocked transition in its present or postset. Translating this definition in boolean logic, if one such transition is *true*, the place has also to be set to *true*; if all the connected transitions are *false*, the place has to be set to *false*; if some transitions have no value assigned yet, the place remains unset. Since a configuration is defined over the transitions of a net, we have to derive the values of the places. We do that by imposing that each transition set to *true* implies to *true* all the places in its preset and in its postset. Formally: $\bigwedge_{t \in T^c} [\ t \Rightarrow \bigwedge_{p \in \bullet t} p\ \wedge \bigwedge_{p \in t \bullet} p]$.[2]

Assuming the original net is a *WF*-net, to guarantee the configured net is still a *WF*-net, we have to ensure that each node that remains in the configured net be on a directed path from p_I to p_O. This is the only requirement of *WF*-net to be verified, as p_I and p_O are part of the configured net by definition. This means all the nodes composing the directed path should not be *false*. For each node, we can decompose this path into two sub-paths: one from p_I to the node in question and the other from the node to p_O, and verify the property over the nodes of each sub-path. However, as per Definition 6, we can restrict the verification to the places of each sub-path, by deriving the places' values from the ones of the transitions. Indeed, if a non-blocked transition has at least one place in its preset on a directed path from p_I and at least one place in its postset on a directed path to p_O, then the transition is on a directed path from p_I to p_O. When searching for such paths we can restrict our analysis to acyclic paths. In fact a cycle always leads back to the same node, but does not provide any valuable progress from p_I to p_O. Formally, we define an acyclic path as follows:

Definition 7 (Acyclic Path). *Let $PN = (P, T, F)$ be a Petri Net:*

- $\phi = \langle n_1, n_2, ..., n_k \rangle$ *is an* acyclic path *of PN such that $(n_i, n_{i+1}) \in F$ for $1 \leq i \leq k-1$ and $i \neq j \Rightarrow n_i \neq n_j$,*
- $\alpha(\phi) = \{n_1, n_2, ..., n_k\}$ *is the alphabet of ϕ,*
- Φ_{PN} *is the set of all acyclic paths of PN;*
- *for all $n \in P \cup T$, $AC_I(n) = \{\phi \in \Phi_{PN} \mid \phi = (p_I, ..., n)\}$ is the set of all acyclic paths from p_I to n,*
- *for all $n \in P \cup T$, $AC_O(n) = \{\phi \in \Phi_{PN} \mid \phi = (n, ..., p_O)\}$ is the set of all acyclic paths from n to p_O.* □

The set of process constraints is called PC and is defined as follows:

Definition 8 (Process Constraint). *Let $N = (P, T, F)$ be a WF-net. Treating each place and each transition of N with a propositional variable, the process constraint $PC(N)$ is a propositional logic formula over these variables, given by the conjunction of the following expressions:*

[2] Where with t, p we indicate a transition, resp. a place, which is set to *true*.

- *p_I and p_O are always true, i.e. $p_I \wedge p_O$;*
- *each place p implies the disjunction of all acyclic paths from p_I to p and the disjunction of all acyclic paths from p to p_O: $\bigwedge_{p \in P} [p \Rightarrow \bigvee_{\phi \in AC_I(p)} (\bigwedge_{n \in \alpha(\phi)} n) \wedge \bigvee_{\phi \in AC_O(p)} (\bigwedge_{n \in \alpha(\phi)} n)]$.* □

The following theorem shows that any configured net derived from a configuration that satisfies PC is a WF-net.

Theorem 1. *Let $N = (P, T, F)$ be a WF-net and $PC(N)$ be its process constraint. Let c be a configuration of N and let $N^c = (P^c, T^c, F^c)$ be the resulting configured net. Let $v \in T \cup P \rightarrow \{true, false\}$ be such that $v(q) = true$ iff $q \in T^c \cup P^c$. Then N^c is a WF-net $\Leftrightarrow v \models PC(N)$.*

Proof. *See Appendix.* □

PC has to be satisfied over a system of variables represented by the nodes of the net, where the values of the transitions are configured by the user and the values of the places are derived automatically. Checking the satisfiability of PC is an NP-complete problem. To overcome this issue, we propose to use a SAT solver[3] based on Shared Binary Decision Diagrams (SBDDs). Existing SBDD solvers can efficiently deal with systems made up of around one million possibilities [15]. Hence they are reasonably adequate to capture all the configurations produced by a reference process model.

We propose to use the solver to obtain a reduced representation of PC in conjunctive normal form, where each variable is initially unset. Then we conjunct this formula with each new transition valuation as provided by the user during the configuration process, and further reduce the formula. In this way we do not recalculate PC for each configuration step. The solver can only reduce the formula if this is satisfiable, i.e. if the configuration can yield a syntactically correct process model. This may imply to automatically force to *true* or *false* the conjunction or disjunction of other transitions we are still unset, in order to keep the formula satisfiable. For example, after blocking t_8 in the model of Fig. 2a, the solver would force to *false* t_5 and t_6 as well.

This solver can be embedded in a tool to support staged configuration of process models, where invalid configurations are identified when a configuration step is applied and alternatives are suggested to keep the model correct.

4.2 Preserving Semantic Correctness

In addition to structural correctness, a configuration should be semantically correct. The example in Fig. 3 shows that a configuration conforming to the WF-net properties is not automatically sound, even if it is derived from a sound WF-net. The WF-net in (a) is a sound WF-net: if t_8 fires before t_4, the token in p_2 can reach p_5 via t_3. However, if t_3 is blocked (b), t_4 needs to fire before t_8 as t_4 depends on the token in p_6 which is removed when t_8 fires. As this behavior is not enforced in the net, the process might deadlock, and is therefore not sound, although all nodes in (b) is still a valid WF-net.

[3] Available at `http://www-verimag.imag.fr/~raymond/tools/bddc-manual`.

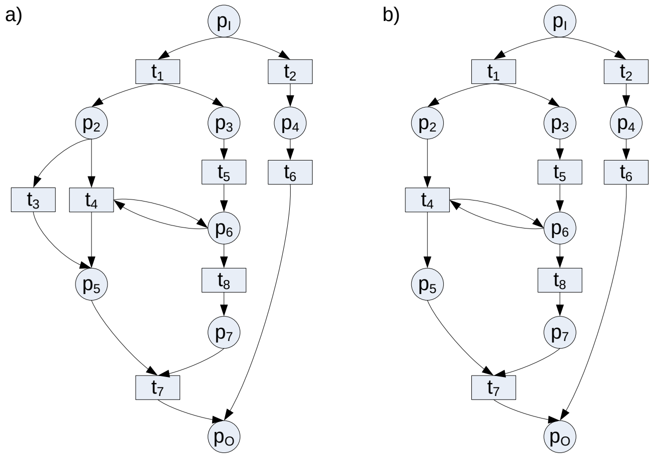

Fig. 3. Blocking t_3 in (a) leads to an unsound WF-net (b)

Soundness is only defined for WF-nets (Definition 2), but it can be generalized to any Petri net with a designated source and sink place. However, it is easy to show that any non WF-net would still violate this generalized soundness notation. Therefore, the process constraint defined in Definition 8 is a necessary requirement for soundness, but as Fig. 3 shows, it is not sufficient.

Below, we prove that PC is a sufficient requirement to guarantee soundness of a configured net, if the original model is a sound extended free-choice WF-net. The restriction to this class of Petri nets provides a good compromise between expressiveness and verification complexity. Not only do extended free-choice WF-nets have several desirable properties [9], but the large majority of constructs of process modeling languages such as EPCs, BPMN or BPEL can be mapped to Petri nets in this class. An extended free-choice is defined as follows [16]:

Definition 9 (Extended Free-choice WF-Net). *Let $N = (P, T, F)$ be a Petri net. N is* extended free-choice *(eFC) if for every couple of places sharing transitions in their postset, these postsets coincide, i.e.* $\forall_{p_1,p_2 \in P\setminus p_O}$ $[p_1\bullet \cap p_2\bullet \neq \varnothing \Rightarrow p_1\bullet = p_2\bullet]$. □

Assuming the reference process model is a sound, *eFC* WF-net, we are able to identify several configuration properties relevant for the preservation of soundness during the configuration process:

Proposition 1 (Properties of Configuration). *Let $N = (P, T, F)$ be a sound, eFC WF-net with source place p_I and sink place p_O, let c be a configuration of N, and let $N^c = (P^c, T^c, F^c)$ be the configured net resulting from c. If N^c is a WF-net (i.e. $PC(N)$ evaluates to* true*), then:*

a) $\forall_{t\in T^c}$ $[(\bullet_N t = \bullet_{N^c} t) \wedge (t\bullet_N = t\bullet_{N^c})]$.
b) $p_I \in P^c$ *and* $p_O \in P^c$.
c) $\forall_{t\in B^c_N}$ $[(\bullet_N t \cap P^c = \varnothing) \vee \exists_{t'\in T^c} (\bullet_N t = \bullet_N t')]$ *(a blocked transition is either not consuming any tokens from P^c or there is a transition in T^c with the same input set).*

d) $\forall_{\sigma \in T^{c*}} \; (M_I \overset{\sigma}{\twoheadrightarrow}_N) \Leftrightarrow (M_I \overset{\sigma}{\twoheadrightarrow}_{N^c})$ *(the input and output sets of transitions in* T^c *are the same in both nets, therefore, the respective behaviors are identical when considering only firing sequences* $\sigma \in T^{c*}$*).*
e) $\forall_{\sigma \in T^{c*}} \; \forall_M \; [(M_I \overset{\sigma}{\twoheadrightarrow}_N M) \Leftrightarrow (M_I \overset{\sigma}{\twoheadrightarrow}_{N^c} M)]$*.*
f) $N^c[M_I\rangle \subseteq N[M_I\rangle$ *(all firing sequences of* N^c *are also possible in* N*).*
g) N^c *is eFC.*
h) $\forall_{M \in N^c[M_I\rangle \setminus \{M_O\}} \; \exists_{t' \in T^c} \; [M[t'\rangle]$ *(*N^c *has no deadlock markings).*

Proof.

a) Follows directly from the construction of N^c*.*
b) Idem.
c) Suppose that some $t \in B^c_N$ *consumes a token from a place* $p \in P^c$ *in* N*. Because* N^c *is a WF-net with source place* p_I *and sink place* p_O*, there has to be a path from* p *to* p_O*. Hence there is a transition* $t' \in T^c$ *consuming a token from* p*. Hence* $\bullet_N t \cap \bullet_N t' \neq \varnothing$*, thus* $\bullet_N t = \bullet_N t'$ *(*N *is eFC).*
d) Follows directly from (a).
e) Follows directly from (d).
f) Follows directly from (e).
g) Let $t, t' \in T^c$ *such that* $\bullet_{N^c} t \cap \bullet_{N^c} t' \neq \varnothing$*. Given that* $\bullet_N t' = \bullet_{N^c} t'$ *and* $\bullet_N t = \bullet_{N^c} t$*, we have* $\bullet_{N^c} t \cap \bullet_{N^c} t' = \bullet_N t \cap \bullet_N t' \neq \varnothing$*. Hence* $\bullet_N t = \bullet_N t'$ *and thus* $\bullet_{N^c} t = \bullet_{N^c} t'$*. Therefore* N^c *is eFC.*
h) Let $M \in N^c[M_I\rangle \setminus \{M_O\}$*. Then using (e) we can deduce* $M_I \twoheadrightarrow_N M$*, thus there exists a* $t \in T$ *such that* $M[t\rangle$ *(as* N *is sound). If* $t \in T^c$ *then we are done. If* $t \in B^c_N$ *then there exists a* $t' \in T^c$ *such that* $\bullet_N t = \bullet_{N^c} t'$ *(c). Therefore* $M[t'\rangle$*.* □

While propositions a, b, d, e and f follow directly from the construction of configured nets and hold for non *eFC WF*-nets, propositions c, g, and h are particularly interesting for soundness. The problem in the example of Fig. 3 is that the configuration may yield an unsound model when a transition is blocked, that shares part of its preset with another transition. By definition, in an *eFC WF*-net such a situation cannot exist and therefore a deadlock marking cannot occur (propositions c and h). Further on, the deadlock in the example prevents all tokens from reaching the final place. As the configured net derived from an *eFC WF*-net remains *eFC* (proposition g), the *eFC* property prevents also this problem as it permits any token to move towards the final place.

These properties allow us to prove that if a configured net, derived from a sound *eFC WF*-net, is a *WF*-net, it fulfills the soundness criteria. Formally:

Theorem 2. *Let* $N = (P, T, F)$ *be a sound, eFC WF-net with source place* p_I *and sink place* p_O*, let* c *be a configuration of* N *and let* $N^c = (P^c, T^c, F^c)$ *be the resulting configured net. If* N^c *is a WF-net, then* N^c *is sound.*

Proof. Note that changing a transition into a silent transition (hiding) has no implications for soundness analysis.

- proper completion: *since* $N^c[M_I\rangle \subseteq N[M_I\rangle$ *(Proposition 1f),* M_O *is the only state marking* p_O*.*

- option to complete: *because* N^c *is an eFC WF-net (Proposition 1g), any token can decide to move towards* p_O*. If* p_O *is marked, all other places are empty (*N^c *has proper completion). Hence, marking* M_O *can be reached (and the property holds) or the net is in a deadlock* M*. However, this is not possible as* N^c *has no deadlock markings (Proposition 1h).*
- no dead transitions: *we define a length function as follows:* $L : T^c \rightarrow \mathbb{N}$*. If* $p_I \in \bullet t$ *then* $L(t) = 0$*. Otherwise* $L(t) = 1 + min_{p\in\bullet t, t'\in\bullet p}\ L(t')$*. Given that every transition in* N^c *is on a path from* p_I*, the function is well-defined. Using induction we prove* $\forall_{n\in\mathbb{N}}\forall_{t\in T^c}\ [L(t) = n \Rightarrow t$ *is not dead in* $N^c]$*.*
  (Base case) If $n = 0$ *then* $\bullet t = \{p_I\}$ *and as* $p_I \in P^c$ *(Proposition 1b),* $M_I[t\rangle$*, hence* t *is not dead.*
  (Induction Hypothesis (IH)) If $t \in T'^c$ *is such that* $L(t) = n+1$*, there exists a transition* t' *such that* $L(t') = n$ *and* $t' \bullet \cap \bullet\, t \neq \emptyset$*.* t' *is not dead (IH), hence there exists an* $M \in N^c[M_I\rangle$ *such that* $M[t'\rangle$*. Let* M' *be such that* $M \xrightarrow{t'} M'$*, then* M' *marks at least one input place (i.e.,* p*) of* t*. As* N^c *has the option to complete,* $M' \twoheadrightarrow M_O$*. This implies that some transition* t'' *exists which removes the token from* p *in some marking* M'*, hence* $p \in \bullet t''$*. Therefore* $\bullet t \cap \bullet t'' \neq \emptyset$*, and thus, given that* N^c *is eFC (Proposition 1g)* $\bullet t = \bullet t''$*. Therefore* $M'[t\rangle$ *and* t *is not dead.* □

Theorems 1 and 2 can be combined to show that a configured net is sound if and only if the process constraint *PC* is satisfied for the corresponding configuration. If the configured net is not an *eFC WF*-net, the implication only holds in one direction and in the other direction soundness cannot be guaranteed. In these cases *PC* can be used to rule out all the syntactically incorrect process models and conventional analysis tools such as Woflan [23] have to be used in addition.

5 Related Work

Variability modeling has been widely studied in the field of Software Product Line Engineering (SPLE) [17]. Techniques developed in the field enable the configuration of software artifacts based on models that relate these artifacts to domain concepts (e.g. parameters, options or features). The techniques differ in the way domain models are captured and related to software artifacts, and also in the way they capture constraints. The Adele Configuration Manager [10] and the Cosmic Configurable Middleware [22] use first-order logic to capture constraints. In contrast, we use propositional logic, for which we can apply efficient techniques to discard incorrect configuration steps or to suggest ways of repairing them. Batory [5] presents a Feature-Oriented Domain Analysis (FODA) technique in which constraints are captured in propositional logic. The respective tool uses a SAT solver to determine if a configuration is valid. A similar approach is adopted in [4]. Our work is inspired by these approaches but it is targeted at business process model configuration. Thus, we deal with graph-oriented models (hence, structural correctness needs special attention) and we are concerned with ensuring absence of deadlocks or livelocks and other behavioral properties.

We outlined a technique to derive propositional logic constraints from process models. Similar techniques have been used for analyzing Petri nets [3] and process graphs [19]. However, the constraints we derive are specifically aimed at checking that a configuration step preserves the structural properties of workflow nets.

Our previous work includes the definition of variation mechanisms for existing process modeling languages: EPCs [18], YAWL [13] and SAP WebFlow [12]. In [14] we proposed a framework to link configurable process models to domain models expressed as questionnaires. Finally, the use of the hiding and blocking operators for defining variation points in process models is sketched in [11].

6 Summary and Outlook

We have proposed a framework for staged correctness-preserving configuration of reference process models. In other words, assuming the initial (reference) process model is correct, the individualized process models are also guaranteed to be correct at each stage of the configuration procedure. This is achieved by capturing the syntactic correctness constraints as a propositional logic formula. This formula, in conjunction with another formula capturing the domain constraints, is used to check the correctness-preservation of each configuration step. If a configuration step violates the constraints, a formula is derived to suggest ways of making the configuration step correctness-preserving. A cornerstone of the framework is a proof that, for free-choice process models, the enforcement of these syntactic constraints also ensures the preservation of semantic correctness.

The proposal is framed in the context of Petri net-based process models. Existing mappings from other process modeling notations to Petri nets provide a basis to enhance the applicability of the framework. This will be a direction for future work. Another direction is to provide tool support based on the proposed framework. In previous work [14], we have developed a tool for questionnaire-driven configuration of C-EPC and C-YAWL process models. After adapting the framework to the syntax of these languages, we will be able to extend this tool with the ability to derive and to enforce correctness-preserving constraints.

References

1. W.M.P. van der Aalst. Verification of Workflow Nets. In *Application and Theory of Petri Nets 1997*, volume 1248 of *LNCS*, pages 407–426. Springer, 1997.
2. W.M.P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theoretical Computer Science*, 270(1-2):125–203, January 2002.
3. P.A. Abdulla, S.P. Iyer, and A. Nyln. SAT-solving the coverability problem for Petri nets. *Formal Methods in System Design*, 24(1):25–43, 2004.
4. M. Antkiewicz and K. Czarnecki. FeaturePlugIn: Feature modeling plug-in for Eclipse. In *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, pages 67–72, 2004.
5. D. S. Batory. Feature Models, Grammars, and Propositional Formulas. In *Proceedings of the 6th International Conference on Software Product Lines (SPLC)*, volume 3714 of *LNCS*, pages 7–20, Rennes, France, September 2005. Springer.

6. T. Curran and G. Keller. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Upper Saddle River, 1997.
7. K. Czarnecki and M. Antkiewicz. Mapping Features to Models: A Template Approach Based on Superimposed Variants. In *Proceedings of the 4th Int. Conference on Generative Programming and Component Engineering*, pages 422–437, Tallinn, Estonia, September 2005. Springer.
8. K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. In *Proceedings of the 5th International Conference on Software Product Lines (SPLC)*, pages 266–283, Boston, MA, USA, September 2004. Springer.
9. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.
10. J. Estublier and R. Casallas. The Adele Software Configuration Manager. In *Configuration Management*, pages 99–139. John Wiley & Sons, 1994.
11. F. Gottschalk, W.M.P. van der Aalst, and M.H. Jansen-Vullers. Configurable Process Models – A Foundational Approach. In *Reference Modeling. Efficient Information Systems Design Through Reuse of Information Models*, pages 59–78. Springer, 2007.
12. F. Gottschalk, W.M.P. van der Aalst, and M.H. Jansen-Vullers. SAP WebFlow Made Configurable: Unifying Workflow Templates into a Configurable Model. In *Proceedings of the 5th International Conference on Business Process Management*, pages 262–270. Springer, September 2007.
13. F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and M. La Rosa. Configurable Workflow Models. BETA Working Paper 222, Eindhoven University of Technology, The Netherlands, 2007.
14. M. La Rosa, J. Lux, S. Seidel, M. Dumas, and A. H. M. ter Hofstede. Questionnaire-driven Configuration of Reference Process Models. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 424–438, Trondheim, Norway, June 2007.
15. S. Minato, N. Ishiura, and S. Yajima. Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean function Manipulation. In *Proceedings of the 27th ACM/IEEE Conference on Design Automation*, pages 52–57, 1990.
16. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
17. K. Pohl, G. Böckle, and F. van der Linden. *Software Product-line Engineering – Foundations, Principles and Techniques*. Springer, Berlin, 2005.
18. M. Rosemann and W. M. P van der Aalst. A Configurable Reference Modelling Language. *Information Systems*, 32(1):1–23, 2007.
19. S.W. Sadiq, M.E. Orlowska, and W. Sadiq. Specification and validation of process constraints for flexible workflows. *Information Systems*, 30(5):349–378, 2005.
20. S. Stephens. The Supply Chain Council and the SCOR Reference Model. *Supply Chain Management - An International Journal*, 1(1):9–13, 2001.
21. C. Taylor and C. Probst. Business Process Reference Model Languages: Experiences from BPI Projects. In *Proceedings of INFORMATIK 2003*, pages 259–263. Jahrestagung der Gesellschaft für Informatik e. V. (GI), 2003.
22. E. Turkay, A.S. Gokhale, and B. Natarajan. Addressing the Middleware Configuration Challenges using Model-based Techniques. In *Proceedings of the 42nd ACM Southeast Regional Conference*, pages 166–170, Huntsville AL, 2004. ACM.
23. 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.

Appendix

Proof of Theorem 1 from Section 4.1:

Proof. ($\Rightarrow$) Let N^c be a *WF*-net and let $v \in T \cup P \rightarrow \{\mathit{true}, \mathit{false}\}$ such that $v(n) = \mathit{true}$ iff $n \in T^c \cup P^c$. As $p_I \in P^c$ and $p_O \in P^c$(Definition 6), $v(p_I) = \mathit{true}$ and $v(p_O) = \mathit{true}$, hence $v \models p_I \wedge p_O$. Since N^c is a *WF*-net, for all $p \in P^c$ there exists at least one directed path from p_I to p. Let $\phi \in AC_I(p)$ be such a path, thus for all $n \in \alpha(\phi)$ we have $n \in P^c \cup T^c$, hence $v(n) = \mathit{true}$. Therefore, $v \models \bigwedge_{n \in \alpha(\phi)} n$. Hence, $v \models \bigvee_{\phi \in AC_I(p)}(\bigwedge_{n \in \alpha(\phi)} n)$. Similarly, as there is at least one path from p to p_O, $v \models \bigvee_{\phi \in AC_O(p)}(\bigwedge_{n \in \alpha(\phi)} n)$, hence $v \models \bigvee_{\phi \in AC_I(p)}(\bigwedge_{n \in \alpha(\phi)} n) \ \wedge \ \bigvee_{\phi \in AC_O(p)}(\bigwedge_{n \in \alpha(\phi)} n)$. Thus, for all $p \in P^c$ $v \models \bigvee_{\phi \in AC_I(p)}(\bigwedge_{n \in \alpha(\phi)} n) \wedge \bigvee_{\phi \in AC_O(p)}(\bigwedge_{n \in \alpha(\phi)} n)$ and therefore for all $p \in P^c$ $v \models p \Rightarrow \bigvee_{\phi \in AC_I(p)}(\bigwedge_{n \in \alpha(\phi)} n) \ \wedge \ \bigvee_{\phi \in AC_O(p)}(\bigwedge_{n \in \alpha(\phi)} n)$. If $p \in P \setminus P^c$ then $v(p) = \mathit{false}$ and thus $v \models p \Rightarrow \bigvee_{\phi \in AC_I(p)}(\bigwedge_{n \in \alpha(\phi)} n) \wedge \bigvee_{\phi \in AC_O(p)}(\bigwedge_{n \in \alpha(\phi)} n)$. Hence $v \models \bigwedge_{p \in P} \ [p \Rightarrow \bigvee_{\phi \in AC_I(p)}(\bigwedge_{n \in \alpha(\phi)} n) \ \wedge \ \bigvee_{\phi \in AC_O(p)}(\bigwedge_{n \in \alpha(\phi)} n)]$.

($\Leftarrow$) Let $v \models PC(N)$. Assume N^c is not a *WF*-net. Since p_I and p_O belong to N^c by definition, choose $p \in P^c$ such that there is either (1) no path from p_I to p or (2) no path from p to p_O. If (1) then for all $\phi \in AC_I(p)$ there is a node $n \in \alpha(\phi)$ such that $n \notin P^c \cup T^c$ and thus $v(n) = \mathit{false}$, $v \not\models n$ and hence for all $\phi \in AC_I(p)$ $v \not\models \bigwedge_{n' \in \alpha(\phi)} n'$ and thus $v \not\models \bigvee_{\phi \in AC_I(p)}(\bigwedge_{n' \in \alpha(\phi) n'})$. If (2) then for all $\phi \in AC_O(p)$ there is a node $n \in \alpha(\phi)$ such that $n \notin P^c \cup T^c$ and thus $v(n) = \mathit{false}$, $v \not\models n$ and hence for all $\phi \in AC_O(p)$ $v \not\models \bigwedge_{n' \in \alpha(\phi)} n'$ and thus $v \not\models \bigvee_{\phi \in AC_O(p)}(\bigwedge_{n' \in \alpha(\phi) n'})$. From both cases we can conclude $v \not\models \bigvee_{\phi \in AC_I(p)}(\bigwedge_{n' \in \alpha(\phi)} n') \ \wedge \ \bigvee_{\phi \in AC_O(p)}(\bigwedge_{n' \in \alpha(\phi)} n')$. Given that $v \models p$, $v \not\models p \Rightarrow \bigvee_{\phi \in AC_I(p)}(\bigwedge_{n' \in \alpha(\phi)} n') \ \wedge \ \bigvee_{\phi \in AC_O(p)}(\bigwedge_{n' \in \alpha(\phi)} n')$. This implies that $v \not\models \bigwedge_{p \in P} \ [p \ \Rightarrow \ \bigvee_{\phi \in AC_I(p)}(\bigwedge_{n' \in \alpha(\phi)} n') \ \wedge \ \bigvee_{\phi \in AC_O(p)}(\bigwedge_{n' \in \alpha(\phi)} n')]$, hence $v \not\models PC(N)$ (Contradiction). □