NATIONAL RESEARCH UNIVERSITY HIGHER SCHOOL OF ECONOMICS

As a manuscript

Roman A. Nesterov

DISCOVERING PROCESS MODELS FOR MULTI-AGENT SYSTEMS FROM EVENT LOGS

PhD Dissertation

for the purpose of obtaining academic degree
Doctor of Philosophy in Computer Science

Academic supervisors:

Doctor of Sciences, Professor
Irina A. Lomazova

PhD, Associate professor
Lucia Pomello,
University of Milano-Bicocca

Moscow – 2022

Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»

На правах рукописи

Нестеров Роман Александрович

СИНТЕЗ МОДЕЛЕЙ ПРОЦЕССОВ ДЛЯ МУЛЬТИАГЕНТНЫХ СИСТЕМ ПО ЖУРНАЛАМ СОБЫТИЙ

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата компьютерных наук

Научные руководители:

доктор физико-математических наук,
профессор Ломазова Ирина Александровна

PhD, доцент
Лючия Помелло,
Миланский университет-Бикокка

Москва – 2022

SCUOLA DI DOTTORATO
UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

Dipartimento di / Department of

Informatics, Systems, and Communications (DISCo)

Dottorato di Ricerca in / PhD program Computer Science Ciclo / Cycle XXXIII

Curriculum in (se presente / if it is)

DISCOVERING PROCESS MODELS FOR MULTI-AGENT SYSTEMS FROM EVENT LOGS

Cognome / Surname Nesterov Nome / Name Roman

Matricola / Registration number 836597

Tutore / Tutor: Lucia Pomello, PhD, Associate professor

Cotutore / Co-tutor: Irina Lomazova, Dr., Professor
HSE University, Moscow, Russia
(se presente / if there is one)

Supervisor:
(se presente / if there is one)

Coordinatore / Coordinator: Leonardo Mariani

ANNO ACCADEMICO / ACADEMIC YEAR 2021-2022

Abstract

A process model discovered from an event log of a multi-agent system often does not fully cover certain viewpoints of its architecture. We consider those concerned with the structure of a model explicitly reflecting agent behavior and interactions. The direct discovery from an event log of a multi-agent system may result in an unclear model structure and over-generalizations of agent behavior. We suggest applying a compositional approach that yields architecture-aware process models of multi-agent systems. An event log of a multi-agent system is filtered by the behavior of individual agents. Then, a multi-agent system model is a composition of agent models discovered from filtered logs. We use an intermediate model, called an interface pattern, specifying agent interactions and representing the architecture of a multi-agent system. We design a collection of specific interface patterns modeling typical agent interactions. An interface pattern provides an abstract specification of interactions and has a part corresponding to the behavior of each agent. We use structural transformations based on morphisms to map agent models discovered from filtered logs on the respective parts in an interface pattern. If such a mapping exists, we guarantee that a composition of agent models preserves their soundness. We conduct a series of experiments to evaluate the compositional approach. Experimental results confirm the improvement in the structure of process models discovered using the compositional approach compared to those discovered directly from event logs.

Keywords: Multi-agent systems · Event Logs · Process discovery · Petri nets · Composition · Morphsims · Transformations · Interface patterns

Sommario

Il modello di un processo sintetizzato da un *event log* di un sistema multi-agente spesso non corrisponde completamente ad alcuni aspetti della sua architettura. Consideriamo gli aspetti riguardanti la struttura di un modello che riflettono esplicitamente il comportamento e le interazioni degli agenti. La sintesi diretta da un event log di un sistema multi-agente può portare ad una struttura del modello poco chiara e a generalizzazioni eccessive del comportamento dell'agente. Proponiamo l'applicazione di un approccio composizionale, che produca modelli di processi di sistemi multi-agente *architecture-aware*. Un event log di un sistema multi-agente include i comportamenti dei singoli agenti. Il modello di un sistema multi-agente si può quindi vedere come composizione dei modelli dei singoli agenti sintetizzati dai *log* filtrati. Usiamo un modello intermedio, chiamato *interface pattern*, che specifica le interazioni degli agenti e rappresenta l'architettura di un sistema multi-agente. Progettiamo una serie di *interface patterns* per modellare le interazioni tipiche degli agenti. Un *interface pattern* fornisce una specifica astratta delle interazioni e ha una parte corrispondente al comportamento di ciascun agente. Usiamo trasformazioni strutturali basate su morfismi per mappare i modelli degli agenti scoperti dai *log* filtrati sulle rispettive parti in un interface pattern. Se tale mappatura esiste, garantiamo che la composizione di modelli di agenti ne preserva la correttezza. Conduciamo una serie di esperimenti per valutare l'approccio composizionale. I risultati sperimentali confermano il miglioramento della struttura dei modelli dei processi sintetizzati con l'approccio composizionale rispetto a quelli sintetizzati direttamente dagli event logs.

Contents

List of Figures

List of Tables

Introduction

Modern information systems produce significant amounts of event data, including, for example, transaction logs, message logs, and records of user activity. These data are commonly referred to as *event logs*. They consist of ordered sequences (*traces*) of records on events that occurred. Event logs are used in *process mining* to discover models of real processes [1]. The expected behavior of processes is usually specified manually at the early stages of the information system life cycle. Discovering the real behavior of processes from event logs is an essential task since manually created process models do not reflect amendments introduced during the operation of an information system.

A wide range of algorithms supports the automated discovery of process models from event logs [2]. Process models can be represented in different notations. Process mining extensively uses various classes of Petri nets, heuristic nets, causal nets, and Business Process Models and Notation (BPMN). This dissertation focuses on modeling the *control-flow* of processes. We abstract from data used in the process execution. We choose Petri nets [3] — a formalism widely used to model process behavior. Petri nets are also the basis for other process modeling notations, e. g., specific classes of BPMN models can be transformed to Petri nets [4].

Four *conformance checking* dimensions, namely fitness, precision, generalization, and simplicity, determine the quality of process discovery algorithms [5]. These dimensions are usually estimated in the interval $[0, 1]$. *Fitness* shows the extent to which a discovered process model can execute traces recorded in an event log. A model is said to *perfectly fit* an event log if it can execute all traces in an event log.

Precision evaluates the ratio between the behavior allowed by a discovered process model and not recorded in an event log. A process model with perfect precision can only execute traces from an initial event log. Perfect precision limits the use of a discovered process model since an event log represents just a finite "snapshot" of all possible process executions. A process model should generalize the behavior recorded in an event log, e. g., trace fragments that correspond to cycles should be identified. *Generalization* is dual to precision. The fourth dimension, *simplicity*, captures the structural complexity of a discovered process model, e. g., whether there are redundant nodes.

A record in an event log typically contains the name of an action and several additional attributes specifying the resources required for executing this action. For instance, in the event log with two short traces shown in Table 1, the "Agent" attribute designates who has executed an action: *John*, *Pete*, or *Nick*. The "Timestamp" attribute helps to order activities in a trace.

Table 1: Event log of a multi-agent system

Timestamp	**Action**	**Agent**
	Trace **1**	
30-12-2020:14.45	register request	Pete
05-01-2021:09.34	check ticket	John
07-01-2021:12.12	examine causally	Pete
09-01-2021:10.15	decide	John, Pete
12-01-2021:13.25	pay compensation	Nick
	Trace **2**	
30-12-2020:16.45	register request	Pete
04-01-2021:10.12	examine thoroughly	Pete
06-01-2021:09.34	check ticket	John
09-01-2021:09.19	decide	John, Pete
10-01-2021:12.26	reject request	Nick

John, *Pete*, and *Nick* execute actions independently, e. g., *Pete* registers a request, *John* checks a ticket, or together, e. g., *John* and *Pete* decide whether to pay the

compensation. We say that an event log, where records contain information on agents, registers the behavior of a *multi-agent system*.

Event logs produced by multi-agent information systems require the additional analysis of agent behavior and interactions. Otherwise, a discovered process model will not fully cover certain viewpoints of its *architecture*. The following motivating example briefly demonstrates this problem. A process model discovered from an event log of a multi-agent system may have the relatively high precision, but its unclear structure does not reflect agent behavior and interactions.

Consider the Petri net shown in Fig. 1. Its structure is self-explanatory, i. e., there are two independent agents communicating via four distinguished nodes. Two places, a and b, are used as asynchronous channels to exchange messages. Firstly, Agent 1 sends a message to channel a, and Agent 2 receives it. Secondly, Agent 2 sends a response back to Agent 1 via channel b. Two transitions marked with s correspond to two actions executed simultaneously by Agents 1 and 2.

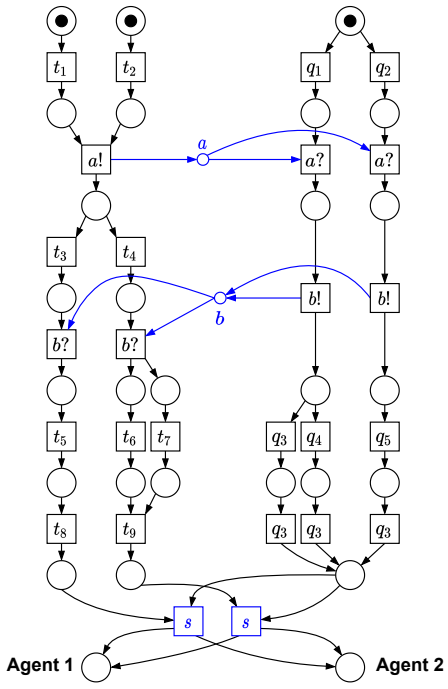


Figure 1: Multi-agent system with two interacting agents

Simulating the behavior of the Petri net from Fig. 1, we generate an event log L of a multi-agent system with two interacting agents. Applying, for example, *Inductive miner* [6] to L, we discover the Petri net shown in Fig. 2. The Inductive miner allows us to guarantee the perfect fitness of a discovered model, i. e., this Petri net can execute every trace in the generated event log L.

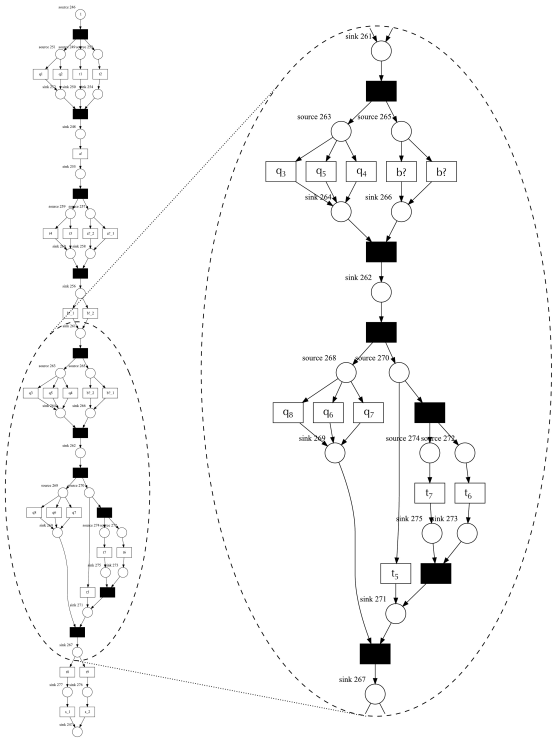


Figure 2: Petri net discovered directly from the event log of the multi-agent system from Fig. 1

In addition, this Petri net demonstrates the high precision evaluation (0.73461). However, the structure of this model is not clear. The Inductive miner has inserted many additional "silent" transitions (black boxes in Fig. 2), which connect blocks of different actions executed independently or together by Agent 1 and 2. The Petri net shown in Fig. 2 does not correctly represent key viewpoints of the architecture of a multi-agent system with two agents exchanging messages.

The direct discovery of a process model from an event log of a multi-agent system can also over-generalize individual agent behavior. For example, in the Petri net shown in Fig. 1, transition q_7 can fire only after transition q_4, whereas in the Petri net shown in Fig. 2, transition q_7 can fire after transitions q_3, q_4, or q_5. The concurrent execution of interacting agents leads to a wide variety of possible traces recorded in an event log. However, a discovered process model should not introduce inappropriate generalizations of individual agent behavior.

The main aim of this dissertation is, given an event log of a multi-agent system, to develop an approach to discovering an *architecture-aware* Petri net, whose structure clearly reflects the architecture of a multi-agent system. In other words, a discovered model should explicitly show agent behavior and interactions, similar to the Petri net shown in Fig. 1, which is discussed above.

Different classes of Petri nets can be used to model the behavior of a multi-agent system. We will use *generalized workflow* (GWF) *nets* that are equipped with initial and final states. They differ from classical workflow nets [7] in allowing initial and final states to be sets of places rather than singletons. For instance, the Petri net shown in Fig. 1 is a GWF-net with three initial and two final places, while the behavior of Agent 2 is a classical workflow net.

This work focuses not only on the structural features of discovered GWF-nets but also on their behavioral properties. *Soundness* [8] is the fundamental correctness property of process behavior. Soundness is also referred to as *proper termination*. A sound process can reach its final state from all intermediate states. The final state of a sound process cannot be contained in any other reachable state. Apart from that, a sound process has no dead actions, which cannot be executed.

Thus, the *purpose* of this work is defined more precisely as follows. Given an event log of a multi-agent system and a specification of agent interactions, the task is to discover a *sound* and an *architecture-aware* GWF-net, such that there are subnets corresponding to agent behavior as well as distinguished nodes corresponding to agent interactions.

A specification of agent interactions is called an *interface*. It represents the key interaction-oriented viewpoints of the architecture of a multi-agent system. We suppose that an interface is provided by experts in advance, e. g., system architects may construct candidate interfaces. The adequacy of these candidates can be determined by checking their conformance to an event log. The discovery of an interface model directly from an event log of a multi-agent system is out of the scope of this dissertation.

Therefore the discovery of process models from event logs of multi-agent systems is based on the following *assumptions*:

1. All records in an event log have the corresponding "Agent" attribute.
2. There is a distinguished set of actions through which agents communicate via message exchange and synchronizations. For instance, in the event log shown in Table 1, the "decide" action is executed by John and Pete together.

We propose a *compositional* approach that allows us to discover sound and architecture-aware process models of multi-agent systems. Even a simple composition of sound process models might not be sound, e. g., it can have a deadlock. That is why we do not consider arbitrary interfaces. The main idea of our solution is to choose specific *interface patterns*, which preserve agent soundness, and to formulate the conditions for a correct application of these patterns. Similar *service interaction patterns* are used in Business Process Management (BPM) for a correct organization of communication in large-scale information systems [9]. Service interaction patterns represent *typical* communication scenarios. We use them to design our collection of interface patterns. An interface pattern is a GWF-net that:

- provides a highly *abstract* view of agent interactions without exposing the internal agent behavior;
- has a part representing the behavior of each agent.

The *central hypothesis* of our study is that using the compositional approach improves the quality of discovered process models in comparison with the quality of process models discovered directly from event logs of multi-agent systems.

Figure 3 shows the three main steps of the compositional approach to discovering process models from event logs of multi-agent systems. These steps are discussed below in detail.

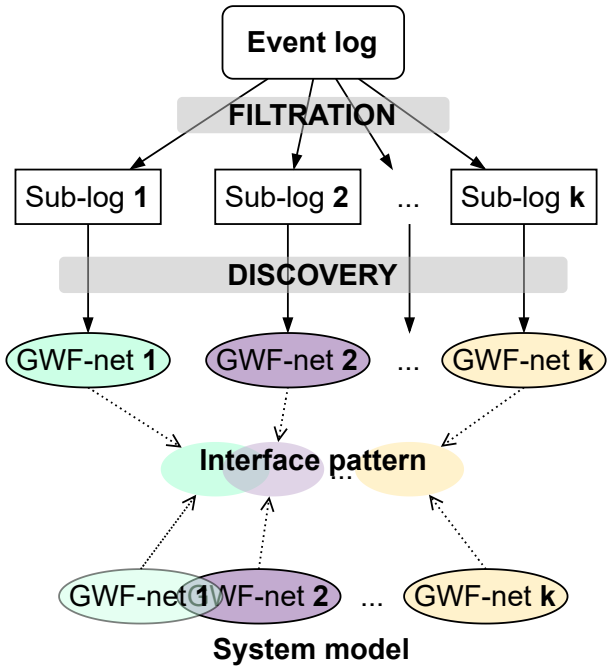


Figure 3: Compositional process discovery

FILTRATION. An event log of a multi-agent system is filtered by actions executed by different agents. Correspondingly, we construct a set of *sub-logs*. For instance, filtering the records in the event log given in Table 1 by the "Pete" value of the "Agent" attribute, we obtain the sub-log presented in Table 2.

DISCOVERY. We discover agent GWF-nets from corresponding sub-logs constructed at the filtration step. At this step, the existing process discovery algorithms can be used. Discovered GWF-nets should be sound. The *Inductive miner*, mentioned above, discovers sound models.

COMPOSITION. If there exists a mapping of an agent GWF-net to the corresponding part in an interface pattern (dashed arcs in Fig. 3), then we can replace this abstract part with the agent GWF-net. As a result, we obtain a sound process model of a multi-agent system, provided that we manage to find a mapping for *every* agent GWF-net. The structure of a resulting model reflects that of an interface pattern, which is represented in Fig. 3 by the overlap between GWF-net 1 and GWF-net 2 matching with the overlap between the corresponding parts in an interface pattern.

Table 2: Pete's sub-log of the event log from Table 1

Timestamp	**Action**	**Agent**
	Trace **1**	
30-12-2020:14.45	register request	Pete
07-01-2021:12.12	examine causally	Pete
09-01-2021:10.15	decide	Pete
	Trace **2**	
30-12-2020:16.45	register request	Pete
04-01-2021:10.12	examine thoroughly	Pete
09-01-2021:09.19	decide	Pete

According to the main scheme of the compositional approach, we develop a compositional process discovery algorithm and prove its correctness.

The *scientific novelty* of the compositional approach to discovering process models from event logs of multi-agent systems lies in identifying the behavior of individual agents and in using interface patterns that help achieve the architecture-aware structure of discovered models. The following three aspects determine the *correctnes* of the compositional process discovery algorithm:

1. Formal backgrounds of an event log filtration and a GWF-net composition.
2. A technique to map agent GWF-nets discovered from filtered sub-logs on the corresponding parts in an interface pattern.

3. A collection of interface patterns describing soundness-preserving interactions among agents in a multi-agent system.

This dissertation discusses these correctness aspects step by step.

Main contributions of the dissertation

1. An algorithm for discovering architecture-aware and sound GWF-nets from event logs of multi-agent systems. The correctness of the algorithm is determined by the preservation of perfect fitness and soundness of agent GWF-nets discovered from sub-logs.
2. Definition and semantical properties of an *asynchronous-synchronous* (AS) composition of GWF-nets used for modeling multi-agent systems, individual agent behavior, and interface patterns.
3. Structural and behavioral properties of transformations based on morphisms (structural abstraction/refinement relations between models) used to map agent GWF-nets on the corresponding subnets in an interface pattern.
4. A collection of sound interface patterns modeling typical agent interactions.
5. Experimental evaluation of the compositional process discovery algorithm regarding central hypothesis of the research. Experimental results confirm the improvement in the quality of process models of multi-agent systems discovered by the compositional approach compared to the quality of process models discovered directly from event logs.

Presentation of contributions

The key results of this thesis were presented and discussed at the following international conferences and workshops:

1. Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE-2021, May 2021, Moscow, Russia). Talk: *Generation of Petri Nets Using Structural Property-Preserving Transformations*

2. International Workshop on Petri Nets and Software Engineering (PNSE-2020, June 2020, online). Talk: *Property-Preserving Transformations of Elementary Net Systems Based on Morphisms*
3. Modeling and Analysis of Complex Systems and Processes (MACSPro-2019, March 2019, Wien, Austria). Talk: *Asynchronous Interaction Patterns for Mining Multi-Agent System Models from Event Logs*
4. Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2018, May 2018, Velikiy Novgorod, Russia). Talk: *Simulating Behavior of Multi-Agent Systems with Acyclic Interactions of Agents*
5. International Workshop "Algorithms & Theories for the Analysis of Event Data" (ATAED 2018, June 2018, Bratislava, Slovakia). Talk: *Compositional Discovery of Workflow Nets from Event Logs Using Morphisms*

The intermediate results of the thesis were regularly discussed at the scientific seminar hosted by the *Laboratory of Process-Aware Information Systems* (PAIS Lab) of the Faculty of Computer Science, HSE University.

Publication of contributions

First-tier publications

1. Nesterov R., Bernardinello L., Lomazova I., Pomello L. *Discovering architecture-aware and sound process models of multi-agent systems: a compositional approach* // Software and Systems Modeling. – Springer, 2022. DOI:10.1007/s10270-022-01008-x.

Second-tier publications

1. Bernardinello L., Lomazova I., Nesterov R., Pomello L. *Property-Preserving Transformations of Elementary Net Systems Based on Morphisms* // Transactions on Petri Nets and Other Models of Concurrency XVI (ToPNoC) / ed. by

M. Koutny, D. Moldt, F. Kordon. – Lecture Notes in Computer Science, vol. 13220. – Springer, 2022. – P. 1–23.

2. Nesterov R., Savelyev S. *Generation of Petri Nets Using Structural Property-Preserving Transformations* // Proceedings of the Institute for System Programming of the RAS. – 2018. – Vol. 33, No. 3. – P. 155–170.
3. Nesterov R., Mitsyuk A., Lomazova I. *Simulating Behavior of Multi-Agent Systems with Acyclic Interactions of Agents* // Proceedings of the Institute for System Programming of the RAS. – 2018. – Vol. 30, No. 3. – P. 285–302.
4. Nesterov R., Lomazova I. *Compositional Process Model Synthesis Based on Interface Patterns* // Tools and Methods of Program Analysis (TMPA-2017) / ed. by V. Itsykson, A. Scedrov, V. Zakharov. – Communications in Computer and Information Science, vol. 779. – Springer, Cham, 2018. – P. 151–162.

Other publications

1. Nesterov R., Lomazova I. *Asycnhronous Interaction Patterns for Mining Multi-Agent System Models from Event Logs* // Proceedings of the MACSPro Workshop 2019 / ed. by I. Lomazova, A. Kalenkova, R. Yavorskiy. – CEUR Workshop Proceedings, vol. 2478. – CEUR-WS.org, 2019. – P. 62–73.
2. Bernardinello L., Lomazova I., Nesterov R., Pomello L. *Compositional Discovery of Workflow Nets from Event Logs Using Morphisms* // Proceedings of the International Workshop ATAED-2018 / ed. by W. van der Aalst, R. Bergenthum, J. Carmona. – CEUR Workshop Proceedings, vol. 2115. – CEUR-WS.org, 2018. – P. 23–38.

Outline

The main part of the dissertation is organized as follows. The first two chapters consider the first correctness aspect of the compositional process discovery. Chapter 1 collects definitions of the basic notions: event logs, generalized workflow nets, and morphisms. In Chapter 2, we define a composition of synchronously and asynchronously interacting GWF-nets and study the main properties of this composition used to model interface patterns and multi-agent systems. Chapter 3 describes the second correctness aspect of the compositional process discovery — an approach for a step-wise construction of mappings between agent GWF-nets discovered from filtered logs and corresponding parts in interface patterns. Chapter 4 presents the formalization of the compositional process discovery algorithm, and a collection of typical interface patterns that forms the third correctness aspect of the compositional process discovery. Also, Chapter 4 provides a formal justification of the correctness of the developed algorithm. Chapter 5 presents the results of the experimental evaluation conducted to compare the quality of process models discovered by the compositional process discovery approach with the quality of models discovered using the standard direct process discovery. Related research is discussed separately in closing sections of Chapters 2, 3, and 4.

Chapter 1

Preliminaries

This chapter discusses the first correctness aspect of the compositional process discovery algorithm — theoretical backgrounds of event log filtration and generalized workflow nets. Firstly, we define functions, multisets, and sequences over sets. Secondly, we formalize event logs of multi-agent systems and log projections that contain the behavior of individual agents. Thirdly, a class of *generalized workflow nets*, we aim to discover from event logs of multi-agent systems, is defined. Finally, relevant notions from *category theory* are recalled.

1.1 Background

Let A and B be two sets. A *function* f from A to B is denoted by $f\colon A \to B$ where A is the *domain* of f, and B is the *range* of f. A function f can also be called a *map* (mapping). The domain of f is denoted by $\mathrm{dom}(f)$. The range of f is denoted by $\mathrm{rng}(f)$. A *restriction* of a function f to a subset $A' \subseteq A$ is denoted by $f|_{A'}\colon A' \to B$. A *partial function* from A to B is a function from A' to B where $A' \subseteq A$. A partial function is denoted by $g\colon A \nrightarrow B$. When g is not defined for an element $a \in A$, we write $g(a) = \perp$. A function $f\colon A \to B$ is called:

- *injective* (injection) iff $\forall a_1, a_2 \in A\colon a_1 \neq a_2 \Rightarrow f(a_1) \neq f(a_2)$.

- *surjective* (surjection) iff $\forall b \in B\, \exists a \in A\colon f(a) = b$.
- *bijective* (bijection) iff f is injective and surjective;
- the *identity* mapping iff $\forall a \in A\colon f(a) = a$.

A *multiset* is a generalization of a set that allows for multiple copies of the same element. Let $\mathbb{N}$ be the set of non-negative integers, and S be a set. A function $m\colon S \to \mathbb{N}$ defines a multiset m over S. We write $s \in m$ iff $m(s) > 0$. The set of all multisets over S is denoted by $\mathcal{B}(S)$. The standard set operations are also extended to multisets as follows. Let $m_1, m_2 \in \mathcal{B}(S)$. Then:

1. $m_1 \subseteq m_2$ iff $m_1(s) \leqslant m_2(s)$ for all elements in S;
2. $m' = m_1 \cup m_2$ iff $m'(s) = m_1(s) + m_2(s)$ for all elements in S;
3. $m'' = m_1 \setminus m_2$ iff $m''(s) = \max(m_1(s) - m_2(s), 0)$ for all elements in S.

Let A^+ denote the set of all finite non-empty sequences over A, and $A^* = A^+ \cup \{\varepsilon\}$, where ε is the empty sequence. Then, given $w \in A^*$ and $B \subseteq A$, $w|_B$ is the *projection* of w on B, i. e., $w|_B$ is the sub-sequence of w obtained after removing elements not belonging to B. Suppose $A = \{a_1, a_2, a_3\}$, $B = \{a_2, a_3\} \subseteq A$, and $w = \langle a_3 a_1 a_2 a_1 a_3 a_2 a_1 \rangle \in A^*$. Then $w|_B = \langle a_3 a_2 a_3 a_2 \rangle$.

1.2 Event Logs and Log Projections

An event log contains records of the observable behavior of an information system. Ordered sequences of records are called *traces*. A trace might occur several times in an event log. Thus, an event log is a multiset of traces.

Definition 1: Event log

Let Λ denote the set of all actions. A *trace* is a finite non-empty sequence σ over Λ, i. e., $\sigma \in \Lambda^+$. An *event log* L over Λ is a multiset over Λ^+, i. e., $L \in \mathcal{B}(\Lambda^+)$.

Given an event log L over Λ and a subset $\Lambda' \subseteq \Lambda$ of action, we can project L on Λ' by projecting every trace in L on Λ'. A log projection should contain only non-empty trace projections. In addition, we should take into account the fact that the projections of different traces in L may coincide.

Definition 2: Log projection

Let $L \in \mathcal{B}(\Lambda^+)$ be an event log and $\Lambda' \subseteq \Lambda$. The projection of L on Λ' is an event log, denoted by $L_{\Lambda'} \in \mathcal{B}((\Lambda')^+)$, which contain non-empty projections of traces in L on Λ', where:

1. $\forall \sigma \in L$: $\sigma|_{\Lambda'} \in L_{\Lambda'}$ iff $\sigma|_{\Lambda'} \neq \varepsilon$.
2. $\forall \sigma \in L$: $\sigma|_{\Lambda'} = \sigma' \in L_{\Lambda'}$: $L_{\Lambda'} = \sum L(\sigma)$.

According to the second requirement of Definition 2, we need to sum the frequencies of traces having identical projections.

Actions in an event log of a multi-agent system are assigned agents executing them. Then Λ can be decomposed into k (possibly intersecting) subsets, i. e., $\Lambda = \Lambda_1 \cup \Lambda_2 \cup ... \cup \Lambda_k$ where k is the number of agents in a multi-agent system. Moreover, a distinguished subset $In \subseteq \Lambda$ of actions is used for agent interactions. In is also called a set of *interacting* actions.

The discovery of an individual agent model from an event log of a multi-agent system L involves projecting the traces in L on the corresponding subset of actions Λ_i, i. e., constructing L_{Λ_i} for $i = 1, 2, ..., k$. Log projections are also called *sub-logs*.

1.3 Generalized Workflow (GWF) Nets

Workflow (WF) nets [7] are basic models used in process discovery. A WF-net is a Petri net with the distinguished initial and final place. The execution of a trace in an event log corresponds to the execution of a WF-net from its initial to its final place. For a more convenient representation of multi-agent systems, we generalize

WF-nets allowing sets of initial and final places rather than singletons. Here, we define *generalized workflow nets* and their behavior.

Definition 3: The structure of a net

A net is a triple $N = (P, T, F)$ where P and T are two disjoint sets of *places* and *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*. For every node $x \in P \cup T$:

1. ${}^\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$ is the *preset* of x.
2. $x^\bullet = \{y \in P \cup T \mid (x, y) \in F\}$ is the *postset* of x.
3. ${}^\bullet x^\bullet = {}^\bullet x \cup x^\bullet$ is the *neighborhood* of x.

Graphically, places of a net are shown by circles, transitions — by boxes, and the flow relation — by arcs.

A net is *P-simple* if $\forall p_1, p_2 \in P$: ${}^\bullet p_1 = {}^\bullet p_2$ and $p_1{}^\bullet = p_2{}^\bullet$ implies $p_1 = p_2$. We consider nets without *self-loops*, i. e., $\forall x \in P \cup T$: ${}^\bullet x \cap x^\bullet = \varnothing$, and isolated transitions, i. e., $\forall t \in T$: $|{}^\bullet t| \geqslant 1$ and $|t^\bullet| \geqslant 1$.

The $\bullet$-notation for presets and postsets, introduced in Definition 3, is also extended to sets of nodes. Let $N = (P, T, F)$ be a net, and $Y \subseteq P \cup T$. Then ${}^\bullet Y = \bigcup_{y \in Y} {}^\bullet y$, $Y^\bullet = \bigcup_{y \in Y} y^\bullet$, and ${}^\bullet Y^\bullet = {}^\bullet Y \cup Y^\bullet$. A subnet of N *generated* by Y is denoted by $N(Y) = (P \cap Y, T \cap Y, F \cap (Y \times Y))$. The set ${}^\bigcirc N(Y) = \{y \in Y \mid \exists z \in (P \cup T) \setminus Y$: $(z, y) \in F$ or ${}^\bullet y = \varnothing\}$ is the *input* border, and the set $N(Y)^\bigcirc = \{y \in Y \mid \exists z \in (P \cup T) \setminus Y$: $(y, z) \in F$ or $y^\bullet = \varnothing\}$ is the *output* border of $N(Y)$.

A *marking* (state) m in a net $N = (P, T, F)$ is a multiset over P, i. e., m: $P \to \mathbb{N}$. A marking m is *safe* if $\forall p \in P$: $m(p) \leqslant 1$. A safe marking is a subset of places. A marking m of place p is depicted by placing $m(p)$ black dots (*tokens*) inside p.

Definition 4: Net system

A net system is a quadruple $N = (P, T, F, m_0)$ where (P, T, F) is a net, and m_0: $P \to \mathbb{N}$ is the *initial* marking.

The behavior of a net system is defined by the transition *firing rule*. A marking m in a net $N = (P, T, F)$ *enables* transition $t \in T$, denoted $m[t\rangle$, if ${}^\bullet t \subseteq m$. Enabled transitions may *fire*. Firing t at m evolves N to a new marking $m' = (m \setminus {}^\bullet t) \cup t^\bullet$, denoted briefly by $m[t\rangle m'$.

A sequence $w \in T^*$ is a *firing* sequence in a net system $N = (P, T, F, m_0)$ if $w = \langle t_1 t_2 ... t_n \rangle$ and $m_0[t_1\rangle m_1[t_2\rangle ... m_{n-1}[t_n\rangle m_n$. Then we write $m_0[w\rangle m_n$. The set of all firing sequences in N is denoted by $FS(N)$.

A marking m in $N = (P, T, F, m_0)$ is *reachable* if $\exists w \in FS(N)\colon m_0[w\rangle m$. Every marking can be reached from itself by firing the empty sequence, i. e., $m[\varepsilon\rangle m$. The set of all markings reachable from m is denoted by $[m\rangle$. N is safe if all reachable markings in N are safe.

A *state machine* is a connected net (P, T, F) where $\forall t \in T\colon |{}^\bullet t| = |t^\bullet| = 1$. A subnet of $N = (P, T, F, m_0)$ *generated* by $Y \subseteq P$ and ${}^\bullet Y^\bullet - N(Y \cup {}^\bullet Y^\bullet)$ – is a *sequential component* of N if it is a state machine and has a single token in the initial marking. N is *covered* by sequential components if every place in N belongs to at least one sequential component. Then N is also called *state machine decomposable* (SMD). For instance, an EN system shown in Fig. 4 has two sequential components generated by $C_1 = \{p_1, p_3, p_4, p_7\}$ and ${}^\bullet C_1{}^\bullet$ as well as by $C_2 = \{p_2, p_5, p_6, p_7\}$ and ${}^\bullet C_2{}^\bullet$. Different sequential components in an SMD-EN system can share both places and transitions. When they share a transition, it is natural to say that sequential components *synchronize*.

State machine decomposability is a basic feature "bridging" structural and behavioral properties of net systems, also considered in [10] to be the important feature of workflow nets. It is easy to see that SMD net systems are safe since their initial markings are safe as well [11]. We further work with SMD net systems, unless otherwise stated explicitly. Thus, we omit the term "SMD" in their specifications.

A *deadlock* in a net system $N = (P, T, F, m_0)$ can be interpreted as a poor synchronization of its sequential components. Since reachable markings in net systems

covered by sequential components are contact-free, we can consider only those deadlocks caused by the absence of tokens in some input places of transitions. For instance, Fig. 4 shows two deadlocks $\{p_3, p_6\}$ and $\{p_4, p_5\}$ that are reachable in the same SMD-EN system from the initial marking $\{p_1, p_2\}$. These deadlocks result from the independent resolution of the local conflicts between t_1 and t_2 as well as t_3 and t_4 by two sequential components: the left generated by $C_1 = \{p_1, p_3, p_4, p_7\}$ and ${}^\bullet C_1{}^\bullet$ and the right generated by $C_2 = \{p_2, p_5, p_6, p_7\}$ and ${}^\bullet C_2{}^\bullet$. In addition, if these local conflicts are resolved differently, s. t. transition t_5 (t_6) is enabled, then it is possible to reach the other deadlock $\{p_7\}$ that can be interpreted as the proper final state of the net system from Fig. 4 since $p_7{}^\bullet = \varnothing$.

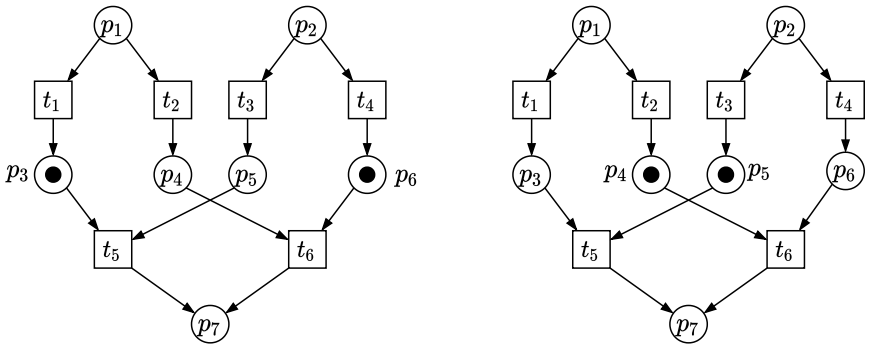


Figure 4: Two deadlocks after the poor synchronization of sequential components

Firing sequences represent the *sequential* behavior of a net system, while its *concurrent* semantics is captured by the *unfolding* [12].

Let $N = (P, T, F)$ be a net, and F^* be the reflexive transitive closure of F. Then for every pair of nodes $x, y \in P \cup T$:

1. x and y are *causally* dependent, denoted $x \leqslant y$, if $(x, y) \in F^*$.
2. x and y are in *conflict*, denoted $x\#y$, if $\exists t_x, t_y \in T\colon t_x \neq t_y, {}^\bullet t_x \cap {}^\bullet t_y = \varnothing$ and $t_x \leqslant x, t_y \leqslant y$.

Definition 5: Occurrence net

A net $O = (B, E, F)$ is an occurrence net if:

1. $\forall b \in B: |{}^\bullet b| \leqslant 1$.
2. F^* is a partial order.
3. $\forall x \in B \cup E: \{y \in B \cup E \mid y < x\}$ is finite.
4. $\forall x, y \in B \cup E: x \# y \Rightarrow x \neq y$.

By Definition 5, an occurrence net O is acyclic. Let $Min(O)$ be the set of nodes in O *minimal* with respect to F^*, i. e., the nodes with the empty preset. Since we consider nets without isolated transitions, $Min(O) \subseteq B$.

Definition 6: Branching process

Let $N = (P, T, F, m_0)$ be a net system, $O = (B, E, F)$ be an occurrence net, and $\pi: B \cup E \to P \cup T$ be a map. A couple (O, π) is a branching process of N if:

1. $\pi(B) \subseteq P$ and $\pi(E) \subseteq T$.
2. $\pi|_{Min(O)}$ is a bijection between $Min(O)$ and m_0.
3. $\forall t \in T: \pi|_{{}^\bullet t}$ is a bijection between ${}^\bullet t$ and ${}^\bullet\pi(t)$.
   Similarly, for $t^\bullet$ and $\pi(t)^\bullet$.
4. $\forall t_1, t_2 \in T:$ if ${}^\bullet t_1 = {}^\bullet t_2$ and $\pi(t_1) = \pi(t_2)$, then $t_1 = t_2$.

The *unfolding* of a net system N, denoted $\mathcal{U}(N)$, is the *maximal* branching process of N, such that any other branching process is isomorphic to a subnet of $\mathcal{U}(N)$, where the map π is restricted to the nodes of this subnet. The map associated with the unfolding is denoted by u and called *folding*.

If a net system contains cycles, its unfolding will be infinite. Figure 5a shows the net system modeling the simple producer-consumer system with the buffer of size 1. The beginning of the unfolding of this net system is provided in Fig. 5b where we also show nodes to which the folding function u maps the nodes in

the unfolding. To overcome the problem of infinite unfoldings, finite complete prefixes were introduced in [13]. In our work, unfoldings will be constructed for acyclic nets. Thus, we do not discuss the formalization of finite prefixes.

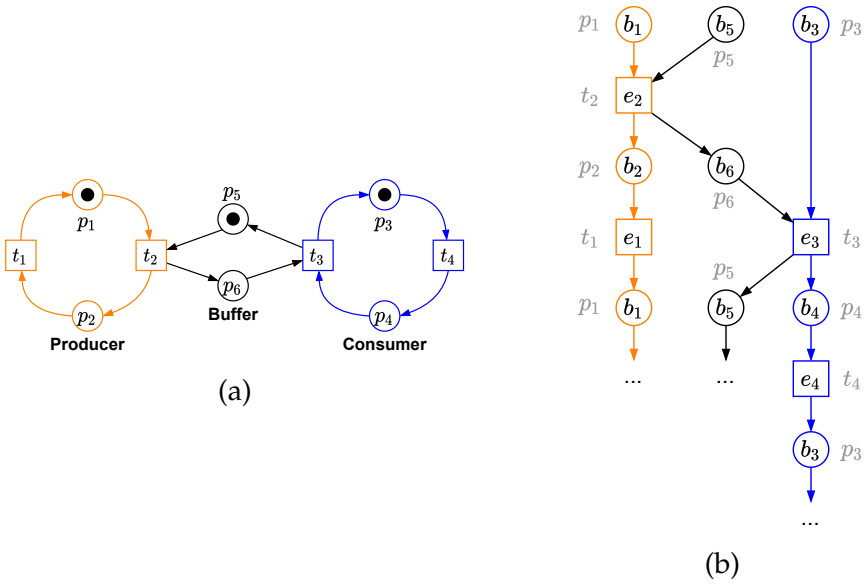


Figure 5: The unfolding of a net system

In a GWF-net, additional restrictions are imposed on the initial marking as well as the *final* marking is distinguished.

Definition 7: Generalized workflow (GWF) net

A generalized workflow net is a net system $N = (P, T, F, m_0)$ equipped with the final marking $m_f \subseteq P$ where:

1. ${}^\bullet m_0 = \varnothing$.
2. $m_f{}^\bullet = \varnothing$.
3. $\forall x \in P \cup T \; \exists s \in m_0 \; \exists f \in m_f \colon (s, x), (x, f) \in F^*$.

According to the third requirement in Definition 7, every node in a GWF-net lies on a path from a place in the initial marking to a place in the final marking.

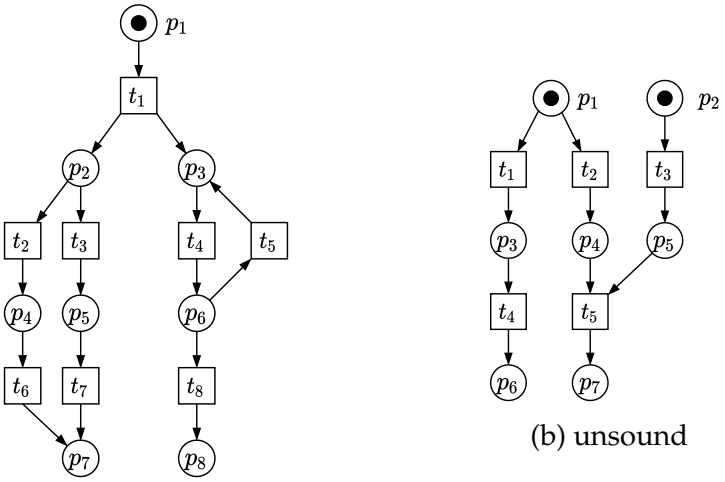
Soundness is the main correctness property of the process behavior represented by a GWF-net. Different kinds of soundness have been studied in [8]. Here, we use the *classical soundness* connected with the reachability of the final marking.

Definition 8: Sound GWF-net

A GWF-net $N = (P, T, F, m_0, m_f)$ is sound if and only if:

1. $\forall m \in [m_0\rangle$: $m_f \in [m\rangle$.
2. $\forall m \in [m_0\rangle$: $m_f \subseteq m \Rightarrow m_f = m$.
3. $\forall t \in T\ \exists m \in [m_0\rangle$: $m[t\rangle$.

In other words, soundness is directly related to the *proper termination* of a corresponding process. Every execution in a properly terminating process must finish in its final state, such that this final state is not contained in any other reachable states. Also, there must not be non-executable (dead) actions.



(a) sound

(b) unsound

Figure 6: Generalized workflow net: two examples

Figure 6 shows two GWF-nets. The first GWF-net shown in Fig. 6a is covered by two sequential components: the one generated by $C_1 = \{p_1, p_2, p_4, p_5, p_7\}$ and ${}^\bullet C_1{}^\bullet$,

and the other generated by $C_2 = \{p_1, p_3, p_6, p_8\}$ and ${}^\bullet C_2{}^\bullet$. This GWF-net is sound according to the requirements imposed by Definition 8. The second GWF-net shown in Fig. 6b is also covered by two sequential components: the one generated by $C_1 = \{p_1, p_3, p_4, p_6, p_7\}$ and ${}^\bullet C_1{}^\bullet$, and the other generated by $C_2 = \{p_2, p_5, p_7\}$ and ${}^\bullet C_2{}^\bullet$. This GWF-net is not sound, since the deadlock $\{p_5, p_6\}$ is reachable from its initial marking $\{p_1, p_2\}$. Thus, its final marking $\{p_6, p_7\}$ is not reachable from all reachable markings, which is the first requirement imposed by Definition 8.

1.4 Categories and Morphisms

Process models of interacting agents discovered from filtered event logs are mapped on the specific parts in an interface pattern by defining morphisms towards the interface pattern. A morphism is a primary tool from *category theory* that studies abstract mathematical objects and relations between these objects. The general definitions of a category, morphisms, based on [14], are given below.

Definition 9: Category, morphisms, properties

A category $\mathcal{C}$ includes a collection of *objects* and a collection of *morphisms* also called arrows. Objects in $\mathcal{C}$ are denoted with capital letters

$$A, B, C, ..., A_1, B_1, C_1, ...$$

and morphisms are denoted with lower-case Latin or Greek letters

$$a, b, c, ..., \alpha, \beta, ...$$

The following is satisfied:

1. Every morphism has the *domain* and *range* among objects in $\mathcal{C}$. A morphism g with the domain A and the range C is denoted $g\colon A \to B$.

2. For every object $C \in \mathcal{C}$, there is a distinguished *identity* morphism $id_C : C \to C$.
3. For every pair of morphisms $f : A \to B$ and $g : B \to C$, s.t. the range of f coincides with the domain of g, the *composition* of f and g is defined as follows: $g \circ f : A \to C$.
4. (*identity* law) If $f : A \to B$, then $id_B \circ f : f$ and $f \circ id_A : f$.
5. (*associativity* law) If $f : A \to B$, $g : B \to C$, and $h : C \to D$, then $h \circ (g \circ f) = (h \circ g) \circ f : A \to D$.

A convenient tool for the visual representation of morphisms is *commutative* diagrams where morphisms are shown with arrows. For instance, the associativity property can be illustrated with the help of the diagram provided in Fig. 7.

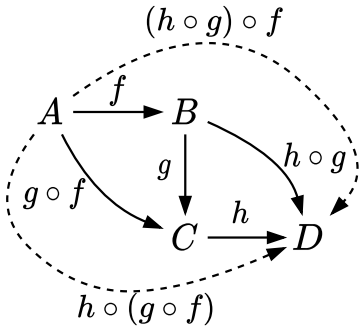


Figure 7: Commutative diagram for the associativity property

1.5 Conclusions of Chapter 1

The first correctness aspect of the compositional approach to discovering process models from event logs of multi-agent systems covers the key theoretical backgrounds.

In this chapter, we collected the basic definitions concerning Petri net theory, introduced generalized workflow nets and recalled the definition of a category, which represents the notion of morphisms in general. Generalized workflow nets are used to model the behavior of individual agents, multi-agent systems, and interface patterns. Generalized workflow nets are convenient for the representation of multi-agent systems, since their initial and final states are allowed to be the sets of places rather than singletons.

In the next chapter, along with defining and studying the semantical properties of a composition of synchronously and asynchronously interacting GWF-nets, we also use certain kinds of morphisms to achieve the preservation of component soundness in this composition.

Chapter 2

Soundness-Preserving Composition

In this chapter, we continue discussing the first correctness aspect of the compositional process discovery algorithm. In this light, we define a composition of interacting GWF-nets used to model an interface pattern and the behavior of a multi-agent system. Firstly, we introduce *transition labels* and a corresponding *Asynchronous-Synchronous* (AS) *composition* that merges synchronous transitions and adds channels between asynchronously interacting transitions in GWF-nets. Secondly, we present a solution to the problem of preserving the soundness of GWF-nets in the AS-composition with the help of an *abstraction/refinement* relation based on morphisms.

2.1 Labeled GWF-Nets: Asynchronous and Synchronous Interactions

An event log L over $\Lambda = \Lambda_1 \cup \Lambda_2 \cup ... \cup \Lambda_k \cup In$, defined in Section 1.2, registers the *observable* behavior of a multi-agent system with k agents. The observable behavior of a GWF-net is derived via transition *labels*. Here, we define *labeled* GWF (LGWF) nets equipped with a total labeling function $h\colon T \to \Lambda \cup \{\tau\}$, where τ is the special label of the *invisible* action. Invisible actions are not recorded in event logs.

A set $\mathfrak{In} \subseteq \Lambda$ contains actions agents interact through. We consider two standard types of interactions among agents in a multi-agent system, namely *asynchronous* and *synchronous* interactions. The behavior of agents is represented via GWF-nets. Below we discuss how the corresponding labels of interacting actions are assigned by $\mathfrak{h}$ to the transitions in a GWF-net.

When GWF-nets interact asynchronously, they exchange messages using *channels*. GWF-nets can *send* (*receive*) messages *to* (*from*) channels. Let $\mathfrak{C} = \{c_1, c_2, ..., c_k\}$ denote the set of all asynchronous channels. Structurally, channels are represented by places. The set $\Theta \subseteq \mathfrak{In}$ of sending/receiving actions implemented with channels is defined as $\Theta = \{c!, c? \mid c \in \mathfrak{C}\}$ where "c!" indicates sending a message to a channel c, and "c?" indicates receiving a message from a channel c. Thus, some transitions in a GWF-net are labeled by asynchronous actions from Θ.

A function $\mathbf{ch}\colon \Theta \to \mathfrak{C}$ maps a sending/receiving action to a corresponding channel, i. e., $\mathbf{ch}(c!) = \mathbf{ch}(c?) = c$. This function is naturally extended to a set of asynchronous actions. If $X \subseteq \Theta$, then $\mathbf{ch}(X) = \bigcup_{\theta \in X} \mathbf{ch}(\theta)$.

A GWF-net may also have transitions with *complement* asynchronous labels — "c!" is complement to "c?" and vice versa. Complement labels are denoted using "overlines", i.e., $\overline{c!} = c?$ and $\overline{c?} = c!$. Then we also require that there exists a place labeled by "c" connecting all transitions labeled by "c!" to all transitions labeled by "c?". Labeled places are necessary to establish the logical dependence between transition with complement labels, i. e., receiving from a channel s should be done only after a message is sent to this channel. However, there can also be other unlabeled places connecting transitions with complement labels.

Synchronous interactions among GWF-nets result in merging transitions corresponding to simultaneous activities. This is formally represented by equal transition labels. By $S = \{s_1, s_2, ..., s_k\} \subseteq \mathfrak{In}$ we denote the set of synchronous actions. Similar to the asynchronous interaction, some transitions in a GWF-net are labeled by synchronous actions from S.

Thus, a set of interacting actions is defined as $\mathfrak{In} = \Theta \cup S$ where $\Theta \cap S = \varnothing$.

Transitions that are not labeled by interacting actions are called *local* since they are not involved in the interaction. Local transitions represent the *internal* behavior of an agent (a multi-agent system).

We formalize the aspects of the asynchronous and synchronous interaction among GWF-nets discussed above in the following Definition 10, where a GWF-net is equipped with two labeling functions. Figure 8 shows the labeled GWF-net, where labeled places are distinguished by the smaller size. By convention, labels are put either inside or near nodes.

Definition 10: Labeled GWF-net

A labeled GWF-net $N = (P, T, F, m_0, m_f, h, k)$ is a GWF-net (P, T, F, m_0, m_f) together with a transition labeling function h and a place labeling function k where:

1. $h\colon T \to \Lambda \cup \{\tau\}$ is a total function.
2. $k\colon P \nrightarrow \mathfrak{C}$ is partial injective function, s. t.:
   (a) $\forall t_1, t_2 \in T$: if $h(t_1) = c!$ and $h(t_2) = c?$, then $\exists p \in P\colon k(p) = c$ and $(t_1, p), (p, t_2) \in F$;
   (b) $\forall p \in P$: if $k(p) = c$, then (${}^\bullet p \neq \varnothing$ where $\forall t \in {}^\bullet p\colon h(t) = c!$) and ($p^\bullet \neq \varnothing$ where $\forall t \in p^\bullet\colon h(t) = c?$).

By Definition 10, it is easy to see that there is the *unique* place labeled by "c" connecting only nonempty sets of transitions with complement labels "c!" and "c?" in an LGWF-net, since function k is injective. This place is also called a *channel*. Other unlabeled places can also connect complement transitions. For example, in Fig. 8, there is the unique place labeled by "h" with the single incoming arc from transition "h!" and the single outgoing arc to transition "h?". However, there is no place labeled by "f" in this LGWF-net since there are no sending transitions labeled by "f!". Two transitions "c!" and "c?" are connected by place c as well as by the unlabeled place with the inverted arc direction (from "c?" to "c!").

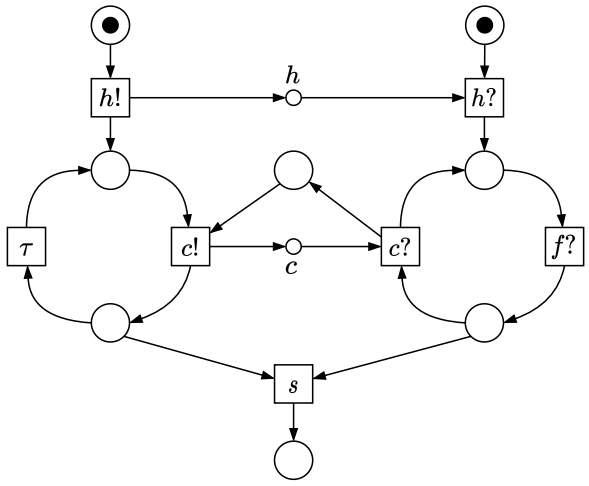


Figure 8: Labeled generalized workflow net

For every firing sequence $w \in \mathsf{FS}(N)$ in an LGWF-net N, we define a labeled *execution* $h(w)$ corresponding to the observable behavior of an LGWF-net.

Definition 11: Execution of LGWF-net

Let $N = (P, T, F, m_0, m_f, h, k)$ be an LGWF-net, and $tw \in \mathsf{FS}(N)$. The corresponding execution $h(tw)$ is defined by the two cases:

1. If $h(t) \neq \tau$, then $h(tw) = h(t)h(w)$.
2. If $h(t) = \tau$, then $h(tw) = h(w)$.

We also observe that the number of transitions with the "c!" label is not less than the number of transitions with the "c?" label in every execution of an LGWF-net, for any place labeled by $c \in \mathfrak{C}$. In other words, the number of times one can receive a message from a channel cannot be greater than the number of times a message has been sent to this channel. This follows from the fact that transition c? can fire only after transition c! if the latter is present in an LGWF-net. There will be the unique labeled place c that is an input place to transition c? and an output place to transition c!.

Let $N^- = (P, T, F, m_0, m_f)$ denote the *underlying* GWF-net obtained from an

LGWF-net $N = (P, T, F, m_0, m_f, h, k)$ by removing labels from transitions and places. An LGWF-net N is sound if the underlying GWF-net N^- is sound.

2.2 AS-Composition of LGWF-nets

Here, an *asynchronous-synchronous* (AS) composition of LGWF-nets is defined. This operation captures synchronous and asynchronous interactions among agents in a multi-agent system according to labels assigned by a labeling function h to transitions in interacting LGWF-nets.

The AS-composition is defined for structurally disjoint LGWF-nets. Intuitively, when composing LGWF-nets, one needs to (1) add and connect channels (labeled places) between transitions with complement asynchronous labels; (2) merge transitions with equal synchronous labels.

The formalization of the AS-composition is given in the following Definition 12 for the basic case of composing two LGWF-nets. It is easy to see that both channel addition and transition synchronization do not lead to the violation of the structural requirements imposed by Definition 7. Thus, in the definition, we explicitly construct an LGWF-net by the AS-composition.

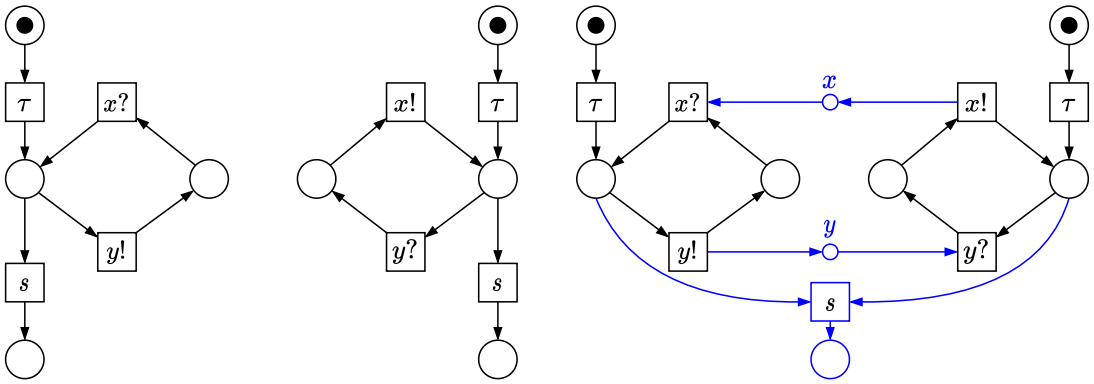
Definition 12: AS-composition of LGWF-nets

Let $N_i = (P_i, T_i, F_i, m_0^i, m_f^i, h_i, k_i)$ be an LGWF-net for $i = 1, 2$, s.t. $(P_1 \cup T_1) \cap (P_2 \cup T_2) = \varnothing$. Let $P_i^u = P_i \setminus \mathrm{dom}(k_i)$ and $T_i^a = \{t_i \in T_i \mid h_i(t_i) \notin S\}$ for $i = 1, 2$. The AS-composition of N_1 and N_2, denoted $N_1 \circledast N_2$, is an LGWF-net $(P, T, F, m_0, m_f, h, k)$ where:

1. $P = P_1^u \cup P_2^u \cup P_c$ where $|P_c| = |C|$,
   $C = \{c \in \mathfrak{C} \mid \exists t, t' \in T_1^a \cup T_2^a : h(t), h(t') \in \Theta, \mathbf{ch}(h(t)) = c, h(t) = \overline{h(t')}\}$.
2. $m_0 = m_0^1 \cup m_0^2$ and $m_f = m_f^1 \cup m_f^2$.
3. $T = T_1^a \cup T_2^a \cup T_{sync}$ where
   $T_{sync} = \{(t_1, t_2) \mid t_1 \in T_1, t_2 \in T_2, h_1(t_1), h_2(t_2) \in S, h_1(t_1) = h_2(t_2)\}$.

4. F is defined by the following cases:
   (a) $\forall p \in P_i^u, \forall t \in T_i^a$ for $i = 1, 2$:
      - $(p, t) \in F \Leftrightarrow (p, t) \in F_i$ and
      - $(t, p) \in F \Leftrightarrow (t, p) \in F_i$.
   (b) $\forall p \in P_1^u, \forall t = (t_1, t_2) \in T_{sync}$:
      - $(p, t) \in F \Leftrightarrow (p, t_1) \in F_1$ and
      - $(t, p) \in F \Leftrightarrow (t_1, p) \in F_1$.
   (c) $\forall p \in P_2^u, \forall t = (t_1, t_2) \in T_{sync}$:
      - $(p, t) \in F \Leftrightarrow (p, t_2) \in F_2$ and
      - $(t, p) \in F \Leftrightarrow (t_2, p) \in F_2$.
   (d) $\forall p \in P_c, \forall t \in T_i^a$ for $i = 1, 2$:
      - $(k(p) = c) \wedge (h_i(t) = c!) \Rightarrow (t, p) \in F$ and
      - $(k(p) = c) \wedge (h_i(t) = c?) \Rightarrow (p, t) \in F$.
5. $h\colon T \to \Lambda \cup \{\tau\}$ where:
   - $\forall t = (t_1, t_2) \in T_{sync}\colon h(t) = h_1(t_1) = h_2(t_2)$ and
   - $\forall t \in T_i^a\colon h(t) = h_i(t)$ for $i = 1, 2$.
6. $k\colon P \nrightarrow C$, s. t. $k|_{P_c}$ is a bijection and $\forall p \notin P_c\colon k(p) = \perp$.

Figure 9b shows the example of composing two LGWF-nets N_1 and N_2 provided in Fig. 9a. Firstly, they exchange messages via channels x and y. Secondly, they synchronize when transitions b and f fire, given by the synchronization label s. As a result, we introduce two labeled places x, y and connect them according to the asynchronous labels of transition pairs c, g and d, h. In addition, we merge transitions b and f obtaining a single transition (b, f) in the AS-composition $N_1 \circledast N_2$ shown in Fig. 9b.



(a) LGWF-nets (b) $N_1 \circledast N_2$

Figure 9: AS-composition of two LGWF-nets

AS-composition of LGWF-nets enjoys several important properties. Firstly, the AS-composition is commutative and associative (see Proposition 1). The proof of these properties is based on a direct construction of the sets of places and transitions according to Definition 12, as the flow relation and the labeling functions are fully characterized by the sets of places and transitions. There is an isomorphism, rather than the equality, since T_{sync} (see Definition 12.3) is the set of ordered transition pairs. Thus, the AS-composition can be generalized to the case of composing more than two LGWF-nets.

Proposition 1: AS-composition is commutative and associative

Let N_1, N_2, N_3 be three LGWF-nets. Then:

1. $N_1 \circledast N_2$ is isomorphic to $N_2 \circledast N_1$.
2. $(N_1 \circledast N_2) \circledast N_3$ is isomorphic to $N_1 \circledast (N_2 \circledast N_3)$.

Secondly, a reachable marking in the AS-composition of LGWF-nets can be "decomposed" into three sub-markings: reachable markings of components together with a marking of labeled places (channels).

Proposition 2: Reachable marking characterization in AS-composition

Let $N_i = (P_i, T_i, F_i, m_0^i, m_f^i, h_i, k_i)$ be an LGWF-net for $i = 1, 2$, and $N_1 \circledast N_2 = (P, T, F, m_0, m_f, h, k)$ be the AS-composition of N_1 and N_2. Then $\forall m \in [m_0\rangle$: $m = (m_1 \backslash \mathrm{dom}(k_1)) \cup (m_2 \backslash \mathrm{dom}(k_2)) \cup m_c$ where $m_1 \in [m_0^1\rangle$, $m_2 \in [m_0^2\rangle$, and $m_c \subseteq \mathrm{dom}(k)$.

The proof of Proposition 2 is based on projecting a firing sequence $w \in FS(N_1 \circledast N_2)$ on the transitions in N_1 and N_2 to obtain the corresponding firing sequences $w_1 = w|_{T_1} \in FS(N_1)$ and $w_2 = w|_{T_2} \in FS(N_2)$ leading to the reachable markings of components, namely $m_1 \in [m_0^1\rangle$, s. t. $m_0^1[w_1\rangle m_1$, and $m_2 \in [m_0^2\rangle$, s. t. $m_0^2[w_2\rangle m_2$.

The AS-composition $N_1 \circledast N_2$ provided in Fig. 10 shows that, in the general case, not every firing sequence of N_1 or N_2 can be found as the projection of a firing sequence in the AS-composition. Here N_2 in isolation has the firing sequence $w = \langle b_1 b_3 b_4 b_3 b_4 b_3 b_4 b_2 \rangle$, while the AS-composition $N_1 \circledast N_2$ only allows firing the cycle b_3–b_4 twice. Indeed, introduction of asynchronous channels, which are labeled places, in $N_1 \circledast N_2$ *restricts* the transition firings possible in the agent LGWF-nets N_1 and N_2.

AS-composition of LGWF-nets may not preserve the soundness of components and their state machine decomposability. For instance, if N_1 and N_2 are two sound LGWF-nets, their composition $N_1 \circledast N_2$ might not be sound. Consider the example shown in Fig. 11 where $N_1 \circledast N_2$ is a result of composing sound LGWF-nets. $N_1 \circledast N_2$ can reach a deadlock $\{f_1, s_2\}$, different from the expected final marking $\{f_1, f_2\}$ (by Definition 12), if N_1 does not send a message to channel d. Thus, $N_1 \circledast N_2$ looses soundness. $N_1 \circledast N_2$ is also no longer covered by sequential components.

The problem of preserving behavioral properties of LGWF-nets in their AS-composition is discussed in the following two sections. Instead of considering the AS-composition of LGWF-nets directly, we analyze an underlying *interface pattern*. An interface pattern is an LGWF-net that models how agents interact (via asynchronous channels and synchronous actions) at the highly abstract level. The

local behavior of agents is almost not represented in an interface pattern.

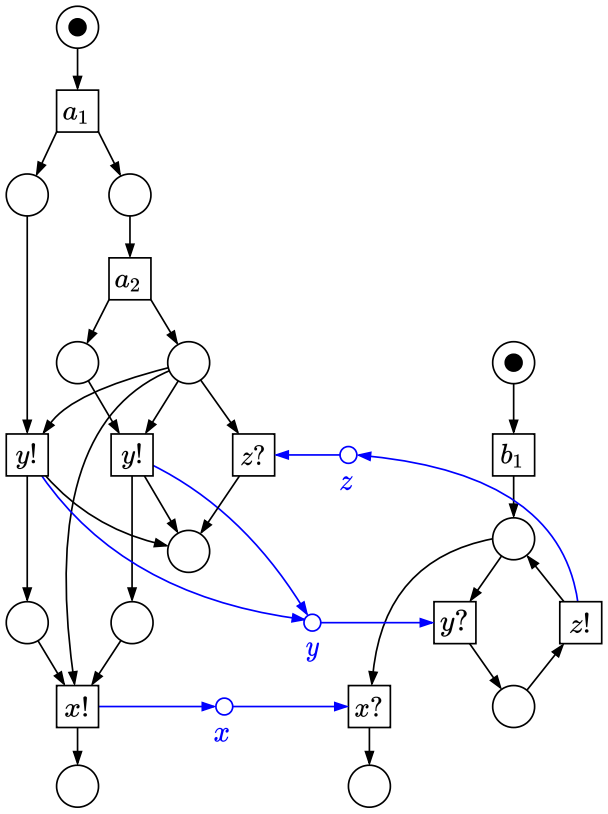


Figure 10: AS-composition $N_1 \circledast N_2$ restricts the behavior of N_1 and N_2

Component and interface LGWF-nets are related via *morphisms* — structural property-preserving relations on net systems — discussed in Section 2.3. Then, in Section 2.4, these morphisms are applied to achieve the preservation of component soundness by the AS-composition.

2.3 Abstraction and Refinement in GWF-nets

This section describes a technique supporting abstraction of subnets and refinement of places in net systems based on α-morphisms. We discuss key properties of

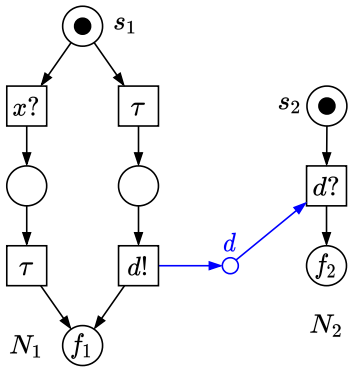


Figure 11: AS-composition may not preserve component properties

α-morphisms relevant to GWF-nets. These properties are further used to address the problem of preserving the soundness of LGWF-nets by their AS-composition.

2.3.1 Place Refinement, Subnet Abstraction, and α-Morphisms

The class of α-morphisms was introduced in [15] to support abstraction and refinement in net systems covered by sequential components. Generalized workflow nets correspond to this class of net systems as well. We consider the definition of α-morphisms on safe net systems and, in the following subsection, discuss the properties of α-morphisms related to GWF-nets.

The example of this morphism is shown in Fig. 12 where N_2 is called an *abstract* net system and N_1 is called a *refinement* of N_2. Refinement of places is depicted by shaded subnets, i. e., subnet $N_1(\varphi^{-1}(p_2))$ in N_1 refines place p_2 in N_2. Refinement of transitions is explicitly given by their names, i. e., two transitions f_1 and f_2 refine the same transition f in N_2. In other words, refinement of places in an abstract net system may lead to splitting its transitions. After recalling the definition of α-morphisms, we also discuss the general intuition behind them.

Definition 13: Relation between two net systems via α-morphism [15]

Let $N_i = (P_i, T_i, F_i, m_0^i)$ be an SMD net system, and $X_i = P_i \cup T_i$ for $i = 1, 2$, s. t. $X_1 \cap X_2 = \varnothing$. An α-morphism from N_1 to N_2 is a total surjective map $\varphi\colon X_1 \to X_2$, also denoted $\varphi\colon N_1 \to N_2$, where:

1. $\varphi(P_1) = P_2$.
2. $\varphi(m_0^1) = m_0^2$.
3. $\forall t_1 \in T_1$: if $\varphi(t_1) \in T_2$, then $\varphi({}^\bullet t_1) = {}^\bullet\varphi(t_1)$ and $\varphi(t_1{}^\bullet) = \varphi(t_1)^\bullet$.
4. $\forall t_1 \in T_1$: if $\varphi(t_1) \in P_2$, then $\varphi({}^\bullet t_1{}^\bullet) = \{\varphi(t_1)\}$.
5. $\forall p_2 \in P_2$:
   (a) $N_1(\varphi^{-1}(p_2))$ is an acyclic net or $\varphi^{-1}(p_2) \subseteq P_1$.
   (b) $\forall p_1 \in {}^\bigcirc N_1(\varphi^{-1}(p_2))$: $\varphi({}^\bullet p_1) \subseteq {}^\bullet p_2$ and if ${}^\bullet p_2 \neq \varnothing$, then ${}^\bullet p_1 \neq \varnothing$.
   (c) $\forall p_1 \in N_1(\varphi^{-1}(p_2))^\bigcirc$: $\varphi(p_1{}^\bullet) = p_2{}^\bullet$.
   (d) $\forall p_1 \in P_1 \cap \varphi^{-1}(p_2)$: $p_1 \notin {}^\bigcirc N_1(\varphi^{-1}(p_2)) \Rightarrow \varphi({}^\bullet p_1) = p_2$ and $p \notin N_1(\varphi^{-1}(p_2))^\bigcirc \Rightarrow \varphi(p_1{}^\bullet) = p_2$.
   (e) $\forall p_1 \in P_1 \cap \varphi^{-1}(p_2)$: there is a sequential component $N' = (P', T', F')$ of N_1, s.t. $p_1 \in P'$, $\varphi^{-1}({}^\bullet p_2{}^\bullet) \subseteq T'$.

By definition, α-morphisms allow us to refine places in N_2 by replacing them with *acyclic* subnets in N_1. If a transition in N_1 is mapped to a transition in N_2, then the neighborhoods of these transitions should correspond (Definition 13.3). If a transition in N_1 is mapped to a place in N_2, then the neighborhood of this transition should be mapped to the same place (Definition 13.4).

The fundamental motivation behind α-morphisms is the possibility to ensure that the behavioral properties of an abstract net system also hold in its refinement. Therefore, every place in the output border of a subnet should have the same choices as a corresponding place in an abstract net system (Definition 13.5c). Places in the input border of a subnet do not need this constraint (Definition 13.5b). Places

in the input border of a subnet cannot be concurrently marked since there are no concurrent transitions in the neighborhood of a subnet (Definition 13.5e). Also, by Definition 13.5d, the neighborhoods of places internal to a subnet are mapped to the same place in an abstract net system as this subnet.

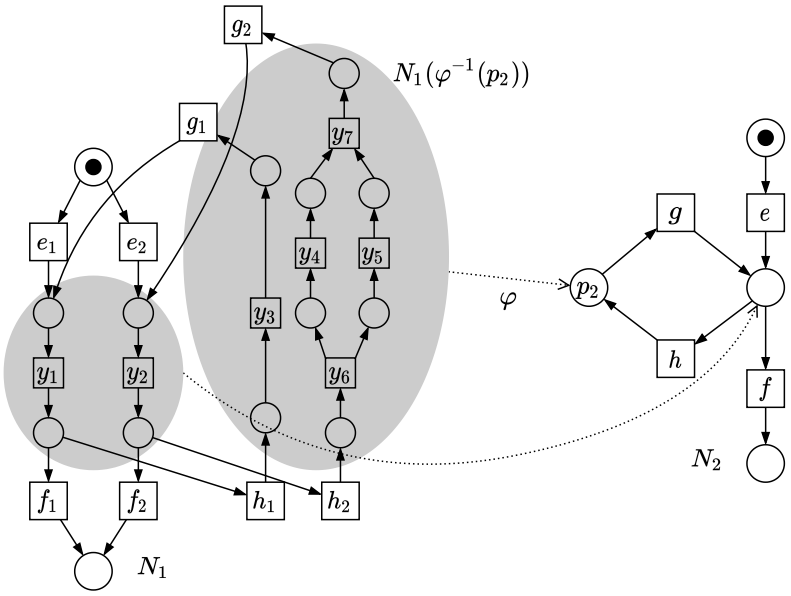


Figure 12: The α-morphism $\varphi\colon N_1 \to N_2$

To sum up, the requirements imposed by Definition 13.5a–5e ensure the main intuition behind α-morphisms. If a subnet in N_1 refines a place in N_2, then this subnet should behave "in the same way" as the abstract place. More precisely, let $N_1(\varphi^{-1}(p_2))$ be a subnet in N_1 refining a place p_2 in N_2 (under $\varphi\colon N_1 \to N_2$). Then the following holds:

1. No tokens are left in $N_1(\varphi^{-1}(p_2))$ after firing transition in $(N_1(\varphi^{-1}(p_2))^{\circ})^{\bullet}$.

2. Transitions ${}^{\bullet}({}^{\circ}N_1(\varphi^{-1}(p_2)))$ are disabled if there is a token in $N_1(\varphi^{-1}(p_2))$.

2.3.2 Properties Preserved and Reflected by α-Morphisms

Here, we discuss properties *preserved* and *reflected* by α-morphisms (see Fig. 13). Several properties were studied in [15]. We will refer to some of them and examine other properties of α-morphisms important for GWF-nets.

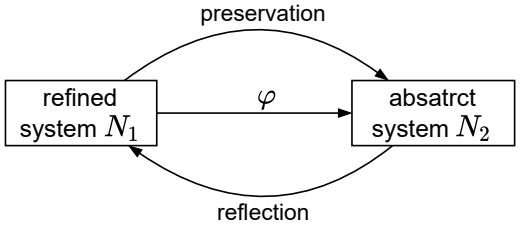


Figure 13: Preservation and reflection of properties under α-morphisms

Proposition 3: Structure of GWF-nets is preserved by α-morphisms

Let $N_i = (P_i, T_i, F_i, m_0^i)$ be an SMD net system, and $X_i = P_i \cup T_i$ for $i = 1, 2$, such that there is an α-morphism $\varphi\colon N_1 \to N_2$. If N_1 is a GWF-net, then N_2 is a GWF-net.

Proof. We show that N_2 satisfies the three structural conditions of GWF-nets imposed by Definition 7.

By Definition 13.2, $\varphi(m_0^1) = m_0^2$. Suppose $\exists p_2 \in m_0^2\colon {}^\bullet p_2 \neq \varnothing$. By Definition 13.5b, $\forall p_1 \in {}^\bigcirc N_1(\varphi^{-1}(p_2))\colon$ if ${}^\bullet p_2 \neq \varnothing$, then ${}^\bullet p_1 \neq \varnothing$. Take $p_1 \in m_0^1$, such that $\varphi(p_1) = p_2$. Since $p_1 \in {}^\bigcirc N_1(\varphi^{-1}(p_2))$, then ${}^\bullet p_1 \neq \varnothing$. By Definition 7.1, $\forall p \in m_0^1\colon {}^\bullet p = \varnothing$. Then, ${}^\bullet p_2 = \varnothing$ and $\forall p \in m_0^2\colon {}^\bullet p = \varnothing$.

By Definition 7.2, $m_f^1 \subseteq P_1$, such that $(m_f^1)^\bullet = \varnothing$. Denote $\varphi(m_f^1)$ by $m_f^2 \subseteq P_2$. Suppose $\exists p_2 \in m_f^2\colon p_2{}^\bullet \neq \varnothing$. Take $p_1 \in m_f^1$, such that $\varphi(p_1) = p_2$. By Definition 13.5c, $\forall p_1 \in N_1(\varphi^{-1}(p_2))^\bigcirc\colon \varphi(p_1{}^\bullet) = p_2{}^\bullet$. Since $p_1 \in N_1(\varphi^{-1}(p_2))^\bigcirc$, $p_1{}^\bullet \neq \varnothing$. But by Definition 7.2, $p_1 \in m_f^1$ and $p_1{}^\bullet = \varnothing$. Then, $p_2{}^\bullet = \varnothing$ and $\forall p \in m_f^2\colon p^\bullet = \varnothing$.

Suppose $\exists x_2 \in X_2$, such that $\forall p \in m_0^2\colon (p, x_2) \notin F_2^*$. Since an α-morphism is a surjective map, $\varphi^{-1}(x_2) \neq \varnothing$. Thus, $\varphi^{-1}(x_2) = \{x_1^1, \ldots, x_1^k\} \subseteq X_1$, where $k \geqslant 1$. If

$x_2 \in T_2$, then $\varphi^{-1}(x_2) \subseteq T_1$, and we take $x_1 \in \varphi^{-1}(x_2)$. If $x_2 \in P_2$, then we take $x_1 \in {}^{\bigcirc}N_1(\varphi^{-1}(x_2))$. By Definition 7.3, $\exists s \in m_0^1 \colon (s, x_1) \in F_1^*$. Then, $\varphi({}^\bullet x_1) \in {}^\bullet x_2$ or $\varphi({}^\bullet x_1) = x_2$. We follow backward the whole path from s to x_1 in N_1 mapping it on N_2 with φ. Thus, we obtain that $\exists x' \in X_2 \colon (x', x_2) \in F_2^*$ and $\varphi(s) = x'$, where $x' \in m_0^2$.

Suppose $\exists x_2 \in X_2$, such that $\forall p \in m_f^2 \colon (x_2, p) \notin F_2^*$. Since an α-morphism is a surjective map, $\varphi^{-1}(x_2) \neq \varnothing$. Thus, $\varphi^{-1}(x_2) = \{x_1^1, \ldots, x_1^k\} \subseteq X_1$, where $k \geqslant 1$. If $x_2 \in T_2$, then $\varphi^{-1}(x_2) \subseteq T_1$, and we take $x_1 \in \varphi^{-1}(x_2)$. If $x_2 \in P_2$, then we take $x_1 \in N_1(\varphi^{-1}(x_2))^{\bigcirc}$. By Definition 7.3, $\exists f \in m_f^1 \colon (x_1, f) \in F_1^*$. Then, $\varphi(x_1{}^\bullet) \in x_2{}^\bullet$ or $\varphi(x_1{}^\bullet) = x_2$. We follow the whole path forward from x_1 to f in N_1 mapping it on N_2 with φ. Thus, we obtain that $\exists x' \in X_2 \colon (x_2, x') \in F_2^*$ and $\varphi(f) = x'$, where $x' \in m_f^2$. □

It also follows from Proposition 3 that $\varphi(m_f^1) = m_f^2$, i. e., the final markings of GWF-nets are preserved by α-morphisms. In the general case, the converse of Proposition 3 is not true. Indeed, as shown in Fig. 14a, α-morphisms may not reflect the initial markings of GWF-nets properly — the inverse image of the initial marking in N_2 is not the initial marking in N_1.

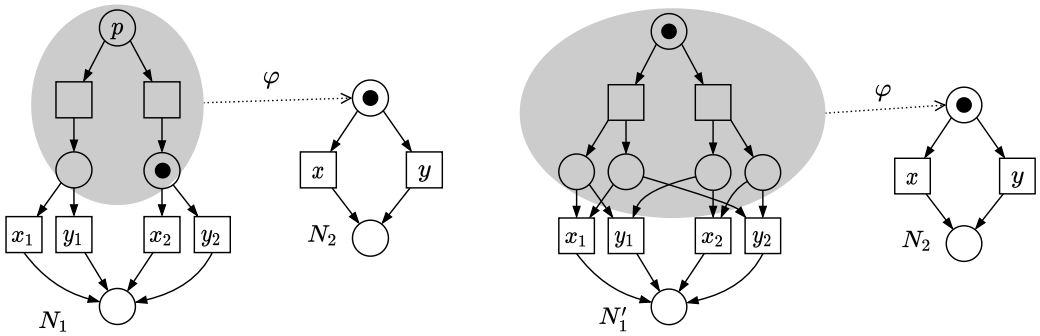


Figure 14: Two α-morphisms with the common range

A refinement N_1 is called *well marked* under $\varphi \colon N_1 \to N_2$ if every place in the input border of a subnet in N_1 refining a marked place in N_2 is marked as well.

Consider again the α-morphism shown in Fig. 14a, the token in the shaded subnet should be placed into p to make N_1 well marked under φ.

Proposition 4: Structure of GWF-nets can be reflected by α-morphisms

Let $N_i = (P_i, T_i, F_i, m_0^i)$ be an SMD net system, and $X_i = P_i \cup T_i$ for $i = 1, 2$, such that there is an α-morphism $\varphi\colon N_1 \to N_2$. If N_2 is a GWF-net and N_1 is well marked under φ, then N_1 is a GWF-net.

Proof. We show that N_1 satisfies the three structural conditions of GWF-nets imposed by Definition 7.

By Definition 7.1, $\forall s_2 \in m_0^2\colon {}^\bullet s_2 = \varnothing$. Since N_1 is well-marked w.r.t. φ, $m_0^1 = \{{}^\bigcirc N_1(\varphi^{-1}(s_2)) \mid s_2 \in m_0^2\}$. Take $s_2 \in m_0^2$ and the corresponding subnet $N_1(\varphi^{-1}(s_2))$. Suppose $\exists p \in {}^\bigcirc N_1(\varphi^{-1}(s_2))$, such that ${}^\bullet p \neq \varnothing$. Then $\varphi(p) = s_2$ and, by Definition 13.5b, $\varphi({}^\bullet p) \subseteq {}^\bullet s_2 = \varnothing$ contradicting the total surjectivity of φ.

By Definition 7.2, $\forall f_2 \in m_f^2\colon f_2{}^\bullet = \varnothing$. Take $f_2 \in m_f^2$ and the corresponding subnet $N_1(\varphi^{-1}(f_2))$. Also take $p \in N_1(\varphi^{-1}(f_2))^\bigcirc$. Then $\varphi(p) = f_2$. By Definition 13.5c, $\varphi(p^\bullet) = f_2{}^\bullet = \varnothing$ contradicting the total surjectivity of φ. Thus, we obtain the final marking of N_1, i.e., $m_f^1 = \{N_1(\varphi^{-1}(f_2))^\bigcirc \mid f_2 \in m_f^2\}$ and $(m_f^1)^\bullet = \varnothing$.

Suppose $\exists x_1 \in X_1$, such that $\forall s_1 \in m_0^1\colon (s_1, x_1) \notin F_1^*$. If $(x_1, x_1) \notin F_1^*$, we follow the path from x_1 to the first node $x_1' \in X_1$ in N_1 backward, such that ${}^\bullet x_1' = \varnothing$. Since $\forall t_1 \in T_1\colon |{}^\bullet t_1| \geqslant 1$, $x_1' \in P_1$. If $x_1' \notin m_0^1$, then N_1 is not well-marked w.r.t. φ. If $(x_1, x_1) \in F_1^*$, then, by Definition 13.5a, there is a corresponding image cycle in N_2. Take $x_2 \in X_2$, such that $\varphi(x_1) = x_2$. By Definition 7.3, $\exists s_2 \in m_0^2\colon (s_2, x_2) \in F_2^*$. Take $x_2' \in X_2$ belonging to this cycle, where at least one node in ${}^\bullet x_2'$ is not in the cycle. By surjectivity of φ, $\exists x_1' \in X_1\colon \varphi(x_1') = x_2'$ belonging to the cycle $(x_1, x_1) \in F_1^*$. If $x_2' \in T_2$, then $\varphi^{-1}(x_2') \subseteq T_1$. By Definition 13.3, the neighborhood of transitions is preserved by φ. Then, $\forall t_1 \in \varphi^{-1}(x_2')\colon \varphi({}^\bullet t_1) = {}^\bullet x_2'$, i.e., there is a place in ${}^\bullet\varphi^{-1}(x_2')$ which does not belong to the cycle $(x_1, x_1) \in F_1^*$. If $x_2' \in P_2$, then take ${}^\bigcirc N_1(\varphi^{-1}(x_2'))$. At least one place in ${}^\bigcirc N_1(\varphi^{-1}(x_2'))$ has an input transition which does not belong to the cycle $(x_1, x_1) \in F_1^*$, since there is a node in ${}^\bullet x_2'$ which is not in the image

cycle in N_2. We have shown that $\exists x \in {}^\bullet x_1'$, such that x does not belong to the cycle $(x_1, x_1) \in F_1^*$. Thus, either there is a path from $\widetilde{x}$ to x with ${}^\bullet\widetilde{x} = \varnothing$, or there is another cycle $(\widetilde{x}, \widetilde{x}) \in F_1^*$.

Applying a similar reasoning, we prove that $\forall x_1 \in X_1\, \exists f_1 \in m_f^1 \colon (x_1, f_1) \in F_1^*$. The only difference is that we follow paths forward. □

In Proposition 3 and 4, we have proven two structural properties of α-morphisms relevant for GWF-nets. We next study the preservation and reflection of behavioral properties, i. e., whether reachable markings are preserved and reflected by α-morphisms.

Proposition 5: Reachable markings are preserved by α-morphisms [15]

Let $N_i = (P_i, T_i, F_i, m_0^i)$ be an SMD net system, and $X_i = P_i \cup T_i$ for $i = 1, 2$, s. t. there is an α-morphism $\varphi \colon N_1 \to N_2$. Let $m_1 \in [m_0^1\rangle$. Then $\varphi(m_1) \in [m_0^2\rangle$. If $m_1[t\rangle m_1'$, where $t \in T_1$, then:

1. $\varphi(t) \in T_2 \Rightarrow \varphi(m_1)[\varphi(t)\rangle\varphi(m_1')$.
2. $\varphi(t) \in P_2 \Rightarrow \varphi(m_1) = \varphi(m_1')$.

In fact, Proposition 5 provides the stronger property, s. t. transition firings are also preserved. In the general case, α-morphisms do not reflect both reachable markings and transition firings. More precisely, given $m_2 \in [m_0^2\rangle$ and $m_2[t_2\rangle$ for a transition $t_2 \in T_2$, it is not possible to say that for all $t \in \varphi^{-1}(t_2)$ there exists $m_1 = \varphi^{-1}(m_2) \in [m_0^1\rangle$, such that $m_1[t_1\rangle$.

Note that reflection of reachable markings is an essential property since we seek to infer the behavioral properties of a refinement from those valid for its abstraction. It is required to check additional *local* constraints based on unfoldings to achieve the reflection of reachable markings. This technique introduced in [15] is briefly described below.

Let N_1 and N_2 be two Petri nets related via the α-morphism $\varphi : N_1 \to N_2$.

Recall that N_1 is called a refinement, and N_2 is called an abstraction of N_1. For every place p_2 in N_2 refined by a subnet in N_1, we construct a *local* net, denoted by $S_2(p_2)$, by taking the neighborhood transitions of p_2 with artificial input and output places if necessary. The same is done for the refined system N_1. We construct the corresponding local net, denoted by $S_1(p_2)$, by taking the subnet in N_1 refining p_2 via φ, i. e., $N_1(\varphi^{-1}(p_2))$ and the transitions $\varphi^{-1}({}^\bullet p_2) \cup \varphi^{-1}(p_2{}^\bullet)$ with artificial input and output places if necessary. We then have two local nets $S_1(p_2)$ and $S_2(p_2)$.

Since there is the α-morphism $\varphi : N_1 \to N_2$, there is also the α-morphism $\varphi^S \colon S_1(p_2) \to S_2(p_2)$ corresponding to the restriction of φ to the places and transitions in $S_1(p_2)$. Recall that the unfolding of a Petri net N, denoted by $\mathcal{U}(N)$, is the maximal branching process of N, such that any other branching process is isomorphic to a subnet in $\mathcal{U}(N)$. The nodes in $\mathcal{U}(N)$ are mapped to the nodes in N via the *folding* function u. In Lemma 1, taking the unfolding of $S_1(p_2)$, we prove that the associated folding function u composed with the α-morphisms φ^S is also the α-morphism under the soundness of N_1. Note that since $S_1(p_2)$ is acyclic (by Definition 13.5a), its unfolding is finite. This helps us to assure that the "final" marking in a subnet in N_1, refining place p_2 in the abstract model N_2, enables exactly the inverse image of transitions in $p_2{}^\bullet$. After proving Lemma 1, we also discuss a specific example of checking these local conditions.

Lemma 1: Soundness is sufficient for local unfolding conditions

Let $N_i = (P_i, T_i, F_i, m_0^i)$ be an SMD net system, and $X_i = P_i \cup T_i$ for $i = 1, 2$, such that there is an α-morphism $\varphi \colon N_1 \to N_2$. Let $\mathcal{U}(S_1(p_2))$ be the unfolding of $S_1(p_2)$ with the folding function u, and φ^S be an α-morphism from $S_1(p_2)$ to $S_2(p_2)$, where $p_2 \in P_2$. Let N_1 be a sound GWF-net. Then, the map from $\mathcal{U}(S_1(p_2))$ to $S_2(p_2)$ obtained as $\varphi^S \circ u$ is an α-morphism.

Proof. Since N_1 is a GWF-net, $S_1(p_2)$ is also a GWF-net. By Lemma 1 in [15], when a transition in $\varphi^{-1}(p_2{}^\bullet)$ fires, it empties the subnet $N_1(\varphi^{-1}(p_2))$. Then $S_1(p_2)$ is sound, and, by Definition 8.3, each transition in $S_1(p_2)$ will occur at least

once. Thus, the folding u is a surjective function from $\mathcal{U}(S_1(p_2))$ to $S_1(p_2)$ and the composition $\varphi^S \circ u$ is the α-morphism from $\mathcal{U}(S_1(p_2))$ to $S_2(p_2)$. □

Figure 15 shows a negative example of checking these local unfolding conditions when N_1 is not sound. It is based on the α-morphism shown in Fig. 14b. In this case, local nets coincides with the original N_1 and N_2. When we unfold N_1, there are no occurrences of transitions y_1 and y_2. Thus, the composition of the corresponding folding function and the α-morphism $\varphi \circ u$ is not an α-morphism. The final marking of the output border in the subnet $N_1(\varphi^{-1}(p_2))$, refining p_2 in N_2, enables transitions x_1 and x_2 only, whereas, in N_2, transition y is also enabled. Therefore, transitions in the inverse image of y in N_2 cannot be enabled by the final markings of the output border in the subnet $N_1(\varphi^{-1}(p_2))$.

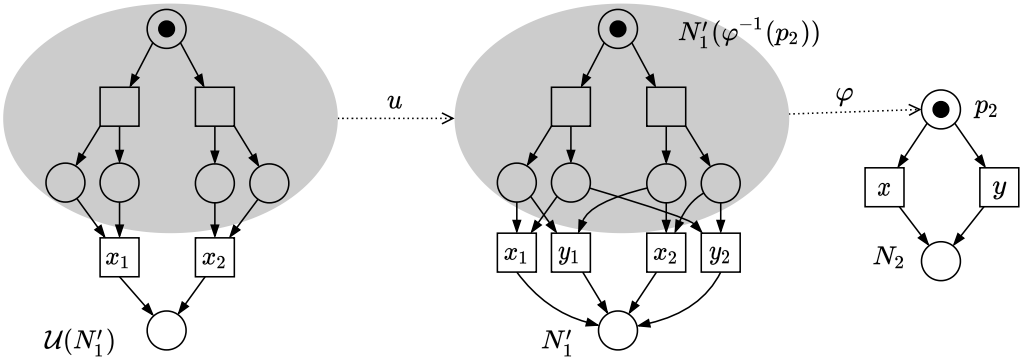


Figure 15: A negative example of checking local unfolding conditions

One should check this unfolding condition for all properly refined places in an abstract model. A properly refined place is a place in an abstract net that is refined by a subnet rather than by a set of places. Taking the above discussion into account, we obtain that α-morphisms reflect reachable markings and transition firings under the soundness of the refinement N_1.

Proposition 6: Soundness is sufficient for reachable marking reflection

Let $N_i = (P_i, T_i, F_i, m_0^i)$ be an SMD net system, and $X_i = P_i \cup T_i$ for $i = 1, 2$, such that there is an α-morphism $\varphi\colon N_1 \to N_2$. Let N_1 be a sound GWF-net. Then $\forall m_2 \in [m_0^2\rangle\, \exists m_1 \in [m_0^1\rangle\colon\ \varphi(m_1) = m_2$. If $m_2[t_2\rangle$, then $\forall t \in \varphi^{-1}(t_2)\, \exists m_1 = \varphi^{-1}(m_2) \in [m_0^2\rangle\colon\ m_1[t_1\rangle$.

Proof of Proposition 6 is based on applying Lemma 1 for all properly refined places in N_1. In the following theorem, we express the key property of α-morphisms concerning the behavioral correctness of GWF-nets.

Theorem 1: Soundness is preserved by α-morphisms

Let $N_i = (P_i, T_i, F_i, m_0^i)$ be an SMD net system, and $X_i = P_i \cup T_i$ for $i = 1, 2$, such that there is an α-morphism $\varphi\colon N_1 \to N_2$. If N_1 is a sound GWF-net, then N_2 is a sound GWF-net.

Proof. We show that N_2 satisfies the three behavioral conditions of a sound GWF-net imposed by Definition 8.

By Definition 8.1, for all $m_1 \in [m_0^1\rangle\colon\ m_f^1 \in [m_1\rangle$. Then, $\exists w \in FS(N_1)\colon\ m_1[w\rangle m_f^1$, i. e., $w = t_1 t_2 \dots t_n$ and $m_1[t_1\rangle m_1^1 \dots m_1^{n-1}[t_n\rangle m_f^1$. Using Proposition 5, it is possible to simulate w in N_2. By Proposition 3, $\varphi(m_f^1) = m_f^2$. Suppose $\exists m_2 \in [m_0^2\rangle\colon\ m_f^2 \notin [m_2\rangle$. By Proposition 6, $\exists m_1' \in [m_0^1\rangle\colon\ \varphi^{-1}(m_2) = m_1'$. By Definition 8.1, $m_f^1 \in [m_1'\rangle$. Thus, $\exists w' \in FS(N_1)\colon\ m_1'[w'\rangle m_f^1$. Using Proposition 5, it is again possible to simulate w' in N_2. Then, $m_f^2 \in [m_2\rangle$.

Suppose $\exists m_2' \in [m_0^2\rangle\colon\ m_2' \supseteq m_f^2$. Then $m_2' = m_f^2 \cup P_2'$, where $P_2' \cap m_f^2 = \varnothing$. By Proposition 6, take $m_1 \in [m_0^1\rangle$, s.t. $\varphi^{-1}(m_1) = m_2'$ and $m_f^1 \not\subseteq m_1$. By Definition 8.1, $m_f^1 \in [m_1\rangle$ and $\exists w \in FS(N_1)\colon\ m_1[w\rangle m_f^1$. Using Proposition 5, it is possible to simulate w in N_2. By Proposition 3, $\varphi(m_f^1) = m_f^2$. The only way to completely empty places in P_2' is to consume at least one token from m_f^2. Then, $\exists f_2 \in m_f^2\colon\ f_2^\bullet \neq \varnothing$ contradicting Definition 7.2.

By Definition 8.3, $\forall t_1 \in T_1\, \exists m_1 \in [m_0^1\rangle\colon m_1[t_1\rangle$. By surjectivity of φ, $\forall t_2 \in T_2\, \exists t_1 \in T_1\colon \varphi(t_1) = t_2$. By Proposition 5, $m_1[t_1\rangle m_1' \Rightarrow \varphi(m_1)[\varphi(t_1)\rangle\varphi(m_1')$. Then, $\forall t_2 \in T_2\, \exists m_2 \in [m_0^2\rangle\colon m_2[t_2\rangle$. □

The converse of Theorem 1 is not true in general. Consider again the example shown in Fig. 14b, where N_2 is sound and N_1 is not sound since transitions y_1 and y_2 cannot fire. Thus, α-morphisms do not reflect soundness following from the fact that reachable markings are also not reflected in the general case.

The *soundness reflection* is an important property of α-morphisms we aim to achieve. In the following section, with the help of α-morphisms, we show when AS-composition preserves the soundness of LGWF-nets. An interface pattern is the AS-composition of abstract LGWF-nets. This composition also represents key interaction-oriented viewpoints of the architecture of a multi-agent system. We develop a technique when the soundness of an interface pattern implies soundness of the AS-composition of refined LGWF-nets. In other words, the corresponding α-morphism from the AS-composition of refined LGWF-nets towards an interface pattern reflects its soundness.

2.4 AS-Composition Can Preserve Soundness

The AS-composition of LGWF-nets preserves component soundness through the use of an intermediate *interface pattern*. This model provides minimal details on the local behavior of communicating agents, focusing on their synchronous and asynchronous interactions. An interface pattern is the AS-composition of corresponding abstract LGWF-nets. Abstraction is implemented using α-morphisms, discussed in the previous section. We need to adjust α-morphisms to LGWF-nets. We aim to deduce the soundness of a refined system model by verifying the soundness of an underlying interface pattern.

Given two LGWF-nets N_1 and N_2, an α-morphism $\varphi\colon N_1 \to N_2$ should additionally respect place and transition labeling, i.e., if a transition in N_1 is mapped to a

transition N_2, then their labels are the same. Specifically, a transition in N_1 labeled with an interacting action from $\mathfrak{h}$ can only be mapped to a transition in N_2 with the same label. The similar requirement should hold for labeled places. This fact implies that only local transitions in N_1 (including invisible ones) can be mapped to a place in N_2. We formalize these restrictions in the following definition.

Definition 14: Restriction of α-morphisms to LGWF-nets

Let $N_i = (P_i, T_i, F_i, m_0^i, m_f^i, h_i, k_i)$ be an LGWF-net, $X_i = P_i \cup T_i$ with $i = 1, 2$. An $\widehat{\alpha}$-morphism from N_1 to N_2 is a total surjective map $\varphi\colon X_1 \to X_2$, also denoted $\varphi\colon N_1 \to N_2$, s. t.:

1′. $\varphi(P_1) = P_2$ where $\forall p_1 \in P_1\colon k_2(\varphi(p_1)) = k_1(p_1)$.

2. $\varphi(m_0^1) = m_0^2$.

2′. $\varphi(m_f^1) = m_f^2$.

3. $\forall t_1 \in T_1\colon$ if $\varphi(t_1) \in T_2$, then $\varphi({}^\bullet t_1) = {}^\bullet\varphi(t_1)$ and $\varphi(t_1{}^\bullet) = \varphi(t_1)^\bullet$.

3′. $\forall t_1 \in T_1\colon$ if $\varphi(t_1) \in T_2$, then $h_2(\varphi(t_1)) = h_1(t_1)$.

3″. $\forall t_1 \in T_1\colon$ if $h_1(t_1) \in I\mathfrak{n}$, then

   (a) $\varphi(t_1) \in T_2$ and

   (b) $h_2(\varphi(t_1)) = h_1(t_1)$.

4. $\forall t_1 \in T_1\colon$ if $\varphi(t_1) \in P_2$, then $\varphi({}^\bullet t_1{}^\bullet) = \{\varphi(t_1)\}$.

5. $\forall p_2 \in P_2\colon$

   (a) $N_1(\varphi^{-1}(p_2))$ is an acyclic net or $\varphi^{-1}(p_2) \subseteq P_1$.

   (b) $\forall p_1 \in {}^\bigcirc N_1(\varphi^{-1}(p_2))\colon \varphi({}^\bullet p_1) \subseteq {}^\bullet p_2$, and if ${}^\bullet p_2 \neq \varnothing$, then ${}^\bullet p_1 \neq \varnothing$.

   (c) $\forall p_1 \in N_1(\varphi^{-1}(p_2))^\bigcirc\colon \varphi(p_1{}^\bullet) = p_2{}^\bullet$.

   (d) $\forall p_1 \in P_1 \cap \varphi^{-1}(p_2)\colon p_1 \notin {}^\bigcirc N_1(\varphi^{-1}(p_2)) \Rightarrow \varphi({}^\bullet p_1) = p_2$ and $p \notin N_1(\varphi^{-1}(p_2))^\bigcirc \Rightarrow \varphi(p_1{}^\bullet) = p_2$.

(e) $\forall p_1 \in P_1 \cap \varphi^{-1}(p_2)$: there is a sequential component $N' = (P', T', F')$ of N_1, s. t. $p_1 \in P'$, $\varphi^{-1}({}^\bullet p_2{}^\bullet) \subseteq T'$.

Thus, an $\widehat{\alpha}$-morphism is an α-morphism (see Definition 13) that also satisfies conditions 1′, 2′, 3′, and 3″ of Definition 14. When two LGWF-nets are related by an $\widehat{\alpha}$-morphism, their underlying GWF-nets are related by an α-morphism. That is why $\widehat{\alpha}$-morphisms inherit the properties of α-morphisms (Proposition 3–6 and Theorem 1), which we will use to prove the soundness preservation in the AS-composition of LGWF-nets.

Moreover, it also follows from Definition 14 that labeled places in LGWF-nets are both preserved and reflected by $\widehat{\alpha}$-morphisms. In other words, the image of a labeled place in N_1 is a labeled place in N_2 as well asl the inverse image of a labeled place in N_2 is a labeled place in N_1. Thus, there is a bijection between the sets of labeled places in two LGWF-nets related by an $\widehat{\alpha}$-morphism.

We next discuss our approach to ensuring that the AS-composition of sound LGWF-nets yields a sound LGWF-net as well.

Given two sound LGWF-nets R_1 and R_2, we aim to be sure that $R_1 \circledast R_2$ is sound. It is possible to compose R_1 and R_2 using Definition 12, but their composition may not be sound, as shown in the previous section (see Fig. 11). A technique described below is applied to get the soundness of $R_1 \circledast R_2$ by construction.

We start with abstracting R_1 and R_2 regarding labeled transitions. Thus, we obtain two abstract LGWF-nets N_1 and N_2, s.t. there is an $\widehat{\alpha}$-morphism $\varphi_i \colon R_i \to N_i$ with $i = 1, 2$. According to Theorem 1, N_1 and N_2 are sound. These abstract models N_1 and N_2 are then composed by adding the same channels as R_1 and R_2 and by synchronizing transitions with the same synchronous labels as R_1 and R_2. Correspondingly, $N_1 \circledast N_2$ is an interface pattern the interacting components R_1 and R_2 agree to follow. Then we *verify* soundness and structural properties of the interface pattern $N_1 \circledast N_2$.

Given the sound interface pattern $N_1 \circledast N_2$ and two $\widehat{\alpha}$-morphisms $\varphi_i \colon R_i \to N_i$

with $i = 1, 2$, we construct two new LGWF-nets $R_1 \circledast N_2$ and $N_1 \circledast R_2$ representing different *intermediate* refined models of the same multi-agent system. In an intermediate refinement, the behavior of one agent is fully specified, while the behavior of the other agent is given only at the highly abstract level. It is easy to verify that these intermediate refinements of the interface pattern $N_1 \circledast N_2$ preserve $\widehat{\alpha}$-morphisms, i. e., there is also an $\widehat{\alpha}$-morphism from $R_1 \circledast N_2$ to $N_1 \circledast N_2$ as well as from $N_1 \circledast R_2$ to $N_1 \circledast N_2$. For instance, an $\widehat{\alpha}$-morphism from $R_1 \circledast N_2$ to $N_1 \circledast N_2$ is constructed from the original $\widehat{\alpha}$-morphism $\varphi_1 \colon R_1 \to N_1$ together with an identity mapping of asynchronously labeled transitions in N_2 and a corresponding mapping of synchronized transitions that can also be refined in R_1. Symmetrically, it is possible to show the construction of an $\widehat{\alpha}$-morphism from $N_1 \circledast R_2$ to $N_1 \circledast N_2$.

In Proposition 7, we additionally claim that an $\widehat{\alpha}$-morphism from an intermediate refined model $R_1 \circledast N_2$ to an interface pattern $N_1 \circledast N_2$ *reflects* connections among asynchronously labeled transitions with channels (labeled places). This reflection follows from the fact that labeled places are both preserved and reflected by $\widehat{\alpha}$-morphisms. We will use this property further in the proof of the main theorem on the soundness preservation in the AS-composition of LGWF-nets.

Proposition 7: Refinement of interface patterns preserves α-morphisms

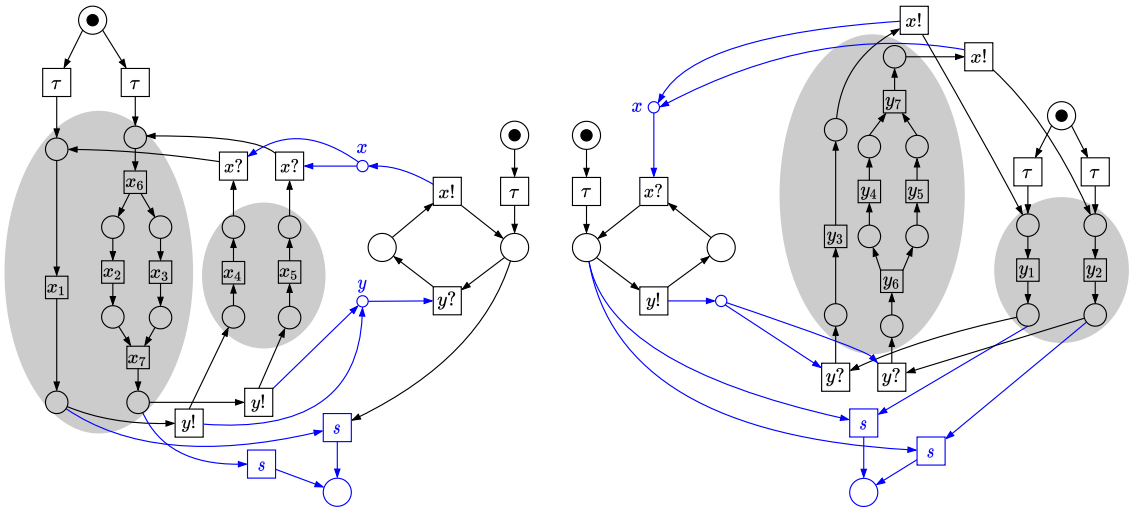
Let R_1, N_1, N_2 be three LGWF-nets, s.t. there is an $\widehat{\alpha}$-morphism $\varphi_1 \colon R_1 \to N_1$. Let $N_1 \circledast N_2 = (P, T, F, m_0, m_f, h, k)$ and $R_1 \circledast N_2 = (P', T', F', m_0', m_f', h', k')$. Then there is an $\widehat{\alpha}$-morphism $\varphi_1' \colon (R_1 \circledast N_2) \to (N_1 \circledast N_2)$, s. t. $\forall p \in \mathrm{dom}(k)$ and $\forall t \in T$:

1. If $(p, t) \in F$, then $\{\varphi^{-1}(p)\} \times \varphi^{-1}(t) \subseteq F'$;
2. If $(t, p) \in F$, then $\varphi^{-1}(t) \times \{\varphi^{-1}(p)\} \subseteq F'$.

Let the AS-composition $N_1 \circledast N_2$ shown in Fig. 9b be an interface pattern. We refine it with two LGWF-nets R_1 and R_2 representing agent behavior and obtain two intermediate refinements shown in Fig. 16. The corresponding $\widehat{\alpha}$-morphisms

are indicated by the shaded subnets. The α-morphism between the underlying GWF-nets R_2^- and N_2^- is provided in Fig. 12, where N_1 corresponds to R_2^-, and N_2 corresponds to N_2^-.

Theorem 2 expresses the main result on the soundness preservation in the AS-composition of LGWF-nets. We prove that an $\widehat{\alpha}$-morphism from an intermediate refinement $R_1 \circledast N_2$ (symmetrically, from $N_1 \circledast R_2$) towards an interface pattern $N_1 \circledast N_2$ *reflects* its soundness. Thus, a multi-agent system model is sound if the interface pattern between agents is sound as well. In proving this fact, we use the properties of α-morphisms discussed in Section and the characterization of reachable markings in the AS-composition (see Proposition 2) together with the reflection property stated in Proposition 7.



(a) AS-composition $R_1 \circledast N_2$ (b) AS-composition $N_1 \circledast R_2$

Figure 16: Two intermediate refinements of $N_1 \circledast N_2$ shown in Fig. 9b

Theorem 2: Intermediate refinement preserves soundness

Let R_1, N_1, N_2 be sound LGWF-nets, s. t. there is an $\widehat{\alpha}$-morphism $\varphi_1 \colon R_1 \to N_1$. If $N_1 \circledast N_2$ is sound, then $R_1 \circledast N_2$ is sound.

Proof. By Proposition 7, there is an $\widehat{\alpha}$-morphism $\varphi_1' \colon (R_1 \circledast N_2) \to (N_1 \circledast N_2)$.

We first fix a notation used in the proof. Let $N_i = (P_i, T_i, F_i, m_0^i, m_f^i, h_i, \ell_i, k_i)$ with $i = 1, 2$, and $R_1 = (\underline{P}_1, \underline{T}_1, \underline{F}_1, \underline{m}_0^1, \underline{m}_f^1, \underline{h}_1, \underline{\ell}_1, \underline{k}_1)$. Also, let $N_1 \circledast N_2 = (P, T, F, m_0, m_f, h, \ell, k)$, and $R_1 \circledast N_2 = (P', T', F', m_0', m_f', h', \ell', k')$.

We show that $R_1 \circledast N_2$ satisfies the three behavioral conditions of a sound LGWF-net imposed by Definition 8.

Take $m' \in [m_0'\rangle$. By Proposition 2 for $R_1 \circledast N_2$, $m' = (\underline{m}_1 \setminus \mathrm{dom}(\underline{k}_1)) \cup (m_2 \setminus \mathrm{dom}(k_2)) \cup m_c$, where $\underline{m}_1 \in [\underline{m}_0^1\rangle$, $m_2 \in [m_0^2\rangle$ and $m_c \in \mathrm{dom}(k')$. By Proposition 5 for φ_1', $\varphi_1'(m') = m \in [m_0\rangle$. By Proposition 2 for $N_1 \circledast N_2$, $m = (m_1 \setminus \mathrm{dom}(k_1)) \cup (m_2 \setminus \mathrm{dom}(k_2)) \cup m_c$, where $m_2 \setminus \mathrm{dom}(k_2)$, m_c are the same as in m', and $m_1 = \varphi_1(\underline{m}_1)$ (by Proposition 5 for φ_1). Since $N_1 \circledast N_2$ is sound, $\exists w \in FS(N_1 \circledast N_2) \colon m[w\rangle m_f$. By Definition 12, recall that $T = T_1^a \cup T_2^a \cup T_{sync}$ in $N_1 \circledast N_2$, where $T_{sync} = \{(t_1, t_2) \mid t_1 \in \mathrm{dom}(\ell_1), t_2 \in \mathrm{dom}(\ell_2), \text{ and } \ell_1(t_1) = \ell_2(t_2)\}$ and $T_i^a = T_i \setminus \mathrm{dom}(\ell_i)$ with $i = 1, 2$. Using interleaving semantics for Petri nets, we can write $w = w_2^1 v$, such that $v = \varepsilon$ or $v = t_1^1 w_s^1 w_2^2 t_1^2 \ldots$, where $w_2^i \in (T_2^a)^*$, $t_1^i \in T_1^a$ and $w_s^i \in T_{sync}^*$ with $i \geqslant 1$. Firstly, each sub-sequence w_2^i can be obviously simulated on the LGWF-net N_2 in $R_1 \circledast N_2$, since φ_1' reflects connections with labeled places (by Proposition 7). Secondly, since R_1 is sound, φ_1 reflects reachable markings and transitions firings (by Proposition 6). Thus, there is a reachable marking $\underline{m}_1^i$ in R_1, belonging to $\varphi_1^{-1}(m_1^i)$ for some $m_1^i \in [m_0^1\rangle$ in N_1. If $m_1^i[t_1^i\rangle$ in N_1, then $\underline{m}_1^i$ enables all transitions in $\varphi_1^{-1}(t_1^i)$ in R_1 as well. Moreover, these transitions are also enabled in $R_1 \circledast N_2$, since φ_1' reflects connections to labeled places (by Proposition 7). Finally, since $N_1 \circledast N_2$ is sound, $\exists m \in [m_0\rangle \colon m[(t_1, t_2)\rangle$ for all (t_1, t_2) in w_s^i. By Proposition 2, $m = m_1 \cup m_2$, where $m_1 \in [m_0^1\rangle$ and $m_2 \in [m_0^2\rangle$ (here $m_c = \varnothing$, since transitions in T_{sync} are not connected with labeled places). Moreover, $m_1[t_1\rangle$ and $m_2[t_2\rangle$. By Proposition 6 for φ_1, there is a reachable marking $\underline{m}_1'$ in R_1, such that $\underline{m}_1' = \varphi_1^{-1}(m_1)$ and $\forall \underline{t}_1 \in \varphi_1^{-1}(t_1) \colon \underline{m}_1'[\underline{t}_1\rangle$. Correspondingly, a reachable marking $\underline{m}_1' \cup m_2$ in $R_1 \circledast N_2$ enables synchronized transitions $(\underline{t}_1, t_2)$ for all $\underline{t}_1 \in \varphi_1^{-1}(t_1)$.

Hence, we reflect the complete firing sequence $w \in FS(N_1 \circledast N_2)$ on $R_1 \circledast N_2$

reaching its final marking m'_f.

Suppose by contradiction $\exists m' \in [m'_0\rangle\colon m' \supseteq m'_f$ and $m' \neq m'_f$. By Definition 12.2, $m'_f = \underline{m}^1_f \cup m^2_f$. Thus, $m' = \underline{m}^1_f \cup m^2_f \cup m_3$. By Proposition 5 for φ'_1, we have that $\varphi'_1(m') \in [m_0\rangle$. Then, $\varphi'_1(m') = \varphi'_1(\underline{m}^1_f) \cup \varphi'_1(m^2_f) \cup \varphi'_1(m_3) = \varphi_1(\underline{m}^1_f) \cup m^2_f \cup m_3 = m^1_f \cup m^2_f \cup m_3 = m_f \cup m_3$. This reachable marking $m_f \cup m_3$ strictly covers the final marking m_f in $N_1 \circledast N_2$ contradicting its soundness.

We show that $\forall t' \in T' \exists m' \in [m'_0\rangle\colon m'[t'\rangle$. By Proposition 2, $m' = (\underline{m}_1 \setminus \mathrm{dom}(\underline{k}_1)) \cup (m_2 \setminus \mathrm{dom}(k_2)) \cup m_c$, where $\underline{m}_1 \in [\underline{m}^1_0\rangle$, $m_2 \in [m^2_0\rangle$ and $m_c \in \mathrm{dom}(k')$. By Definition 12.3, $\forall t' \in T'\colon t' \in \underline{T}^a_1$ or $t' \in T^a_2$ or $t' \in T_{sync}$. If $t' \in T^a_2$, then $\exists m \in [m_0\rangle\colon m[t'\rangle$, since $N_1 \circledast N_2$ is sound. By Proposition 7, $(m_2 \setminus \mathrm{dom}(k_2)) \cup m_c$ in $R_1 \circledast N_2$ also enables t'. If $t' \in \underline{T}^a_1$, then there are two cases. If $\varphi'_1(t') \in P$, then t' is not connected to labeled places. Since R_1 sound, $\underline{m}_1$ enables t'. If $\varphi'_1(t') \in T$, then take $t \in T$, such that $\varphi'_1(t') = t$ (by the surjectivity of φ'_1). Since $N_1 \circledast N_2$ is sound, $\exists m \in [m_0\rangle\colon m[t\rangle$. By Proposition 6 and 7, the reachable marking $\underline{m}_1 \cup m_c$ in $R_1 \circledast N_2$ (being the inverse image of m under φ'_1) enables t'. As for the case when $t' \in T_{sync}$, we have already considered it above when proving reachability of the final marking in $R_1 \circledast N_2$. □

Having two $\widehat{\alpha}$-morphisms from the intermediate refinements $R_1 \circledast N_2$ and $N_1 \circledast R_2$ to the same interface $N_1 \circledast N_2$, we can compose $R_1 \circledast N_2$ and $N_1 \circledast R_2$ using the operation defined in [15]. It is required to (a) substitute subnets in $R_1 \circledast N_2$ and $N_1 \circledast R_2$ for the corresponding places in $N_1 \circledast N_2$; (b) replace transitions in $N_1 \circledast N_2$ with their inverse images merging those with identical images. As a result, we obtain N and two $\widehat{\alpha}$-morphisms from N to $R_1 \circledast N_2$ and $N_1 \circledast R_2$, such that the diagram shown in Fig. 17b commutes, i. e., $\varphi'_1 \circ \varphi''_1 = \varphi'_2 \circ \varphi''_2$, where $\varphi'_1\colon (R_1 \circledast N_2) \to (N_1 \circledast N_2)$, $\varphi'_2\colon (N_1 \circledast R_2) \to (N_1 \circledast N_2)$, $\varphi''_1\colon N \to (R_1 \circledast N_2)$, and $\varphi''_2\colon N \to (N_1 \circledast R_2)$.

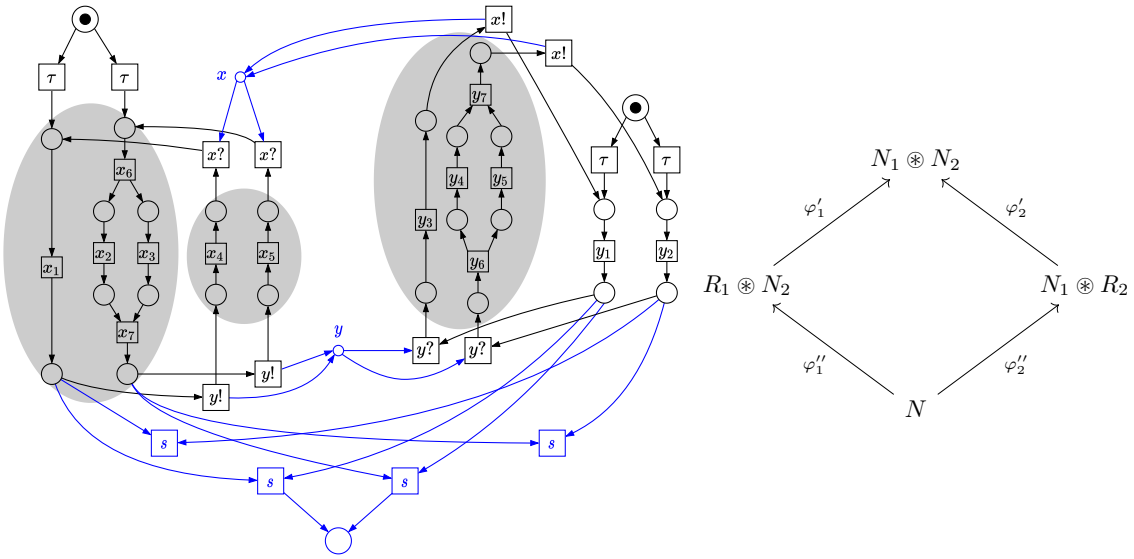
Another way is to construct intermediate refinements again by refining N_2 in $R_1 \circledast N_2$ (N_1 in $N_1 \circledast R_2$). We obtain $R_1 \circledast R_2$, isomorphic to the previously constructed composition N up to renaming of synchronized transitions. According to Proposition 7, there are two $\widehat{\alpha}$-morphisms from $R_1 \circledast R_2$ to $R_1 \circledast N_2$ as well as to

$N_1 \circledast R_2$. According to Theorem 2, since $R_1 \circledast N_2$ ($N_1 \circledast R_2$) is sound, $R_1 \circledast R_2$ is also sound. Therefore, we have also shown that it is possible to simultaneously refine N_1 and N_2 in a sound interface with sound LGWF-nets R_1 and R_2 (see Corollary 1).

In Fig. 17a, we show the result of composing, by means of $\widehat{\alpha}$-morphisms, intermediate refined models $R_1 \circledast N_2$ and $N_1 \circledast R_2$ constructed in Fig. 16. This AS-composition corresponds to the direct AS-composition of R_1 and R_2 as well.

Corollary 1: AS-composition can preserve soundness of components

Let R_i, N_i be sound LGWF-nets, s. t. there is an $\widehat{\alpha}$-morphism $\varphi_i \colon R_i \to N_i$ for $i = 1, 2$. If $N_1 \circledast N_2$ is sound, then $R_1 \circledast R_2$ is sound.



(a) composition N isomorphic to $R_1 \circledast R_2$ (b) diagram

Figure 17: AS-composition of $R_1 \circledast N_2$ and $N_1 \circledast R_2$ (Fig. 16) based on $\widehat{\alpha}$-morphisms

2.5 Related Works: Net System Composition

Petri net composition was extensively studied in the literature. Researchers considered various aspects, including *architectural* concepts of compositional modeling and *semantical* issues relating to the compositional analysis of Petri net behavior. The ubiquity of service-oriented and multi-agent architectures of information retains the relevance of further research on these aspects of Petri net composition.

General frameworks for Petri net composition were discussed in, among the others, [3,16–18]. The recent works [19–21] by W. Reisig are devoted to a systematic study of compositional modeling principles applicable to various formalisms and notations, including (Colored) Petri nets, BPMN (Business Process Modeling and Notation) process models, and UML (Unified Modeling Language) diagrams. He addressed *architectural* problems behind the composition of Petri net components with double-sided interfaces in the context of algebraic properties paying special attention to the associativity.

Several works studied whether a composition of *open* Petri nets preserves semantical properties of components. P. Baldan et al. [22] introduced a class of open Petri nets and analyzed the categorical framework behind open Petri net composition constructed via place and transition fusion. K. van Hee et al. [23, 24] considered a soundness-preserving refinement of places in open WF-nets with sound (composition of) WF-nets.

A class of superposed automata nets (SA-nets) was introduced by F. De Cindo et al. in [25]. SA-nets were among the first formalisms to model systems with synchronously communicating sequential components via transition fusion.

S. Haddad et al. [26] defined the semantics of *input/output* (I/O) Petri nets and their composition constructed through the insertion of asynchronous channels. The authors studied channel properties related to message consumption and interaction termination. It was shown that these properties are decidable and preserved by an asynchronous composition of I/O-Petri nets.

Y. Soussi and G. Memmi [27, 28] considered the problem of liveness preserva-

tion in a composition of Petri nets through an intermediate model of communication medium. Their approach is based on global and rigid structural constraints.

C. Stahl and K. Wolf [29] applied *operating guidelines* for the compositional verification of deadlock-freeness in the composition of open Petri nets. Their work also considered a problem to decide if one can replace a component in a composition preserving its semantical properties. The method proposed by C. Stahl and K. Wolf considered only two parts of the soundness property, namely boundedness and deadlock-freeness. Compositional analysis of the third component — the absence of livelocks — was not addressed.

Inheritance of behavioral properties of Petri nets is also achieved with the help of morphisms — structural property-preserving graph mappings. The composition of Petri nets via morphisms was a subject of many works, including, for example, [30–37]. We note that morphisms provide a natural and rigid framework to explore properties of Petri net composition.

In our study, the soundness preservation in the AS-composition of LGWF-nets is achieved with the help of a restriction of α-morphisms, originally defined by L. Bernardinello et al. in [15]. They allow us to abstract subnets and refine places in Petri nets. In addition, α-morphisms preserve and reflect reachable markings and induce the bisimulation between related models. We extended the applicability of α-morphisms by considering asynchronous interactions among agents in a multi-agent system.

Several works have discussed architectural and semantical aspects of compositional approaches to workflow net modeling. J. Siegeris and A. Zimmermann [38] considered several specific patterns of WF-net interactions preserving the relaxed version of component soundness admitting executions that may not terminate in a final state. The work [39] by I. Lomazova and I. Romanov addressed the problem of preserving service correctness in the context of resources produced and consumed by interacting services. The earlier work [40] by I. Lomazova also proposed an approach to soundness-preserving re-engineering of hierarchical WF-nets with

the two-level structure.

Y. Cardinale et al., in the survey [41], discussed a variety of approaches to the compositional modeling of web services. The authors stressed that there is a lack of service execution techniques based on different classes of Petri nets. In particular, V. Pancratius and W. Stucky [42] considered a composition of WF-nets representing web service behavior with the help of a family of adapted relational algebra operations.

The main difference in our work is that the AS-composition of labeled GWF-nets leaves asynchronous channels and synchronous transitions open for other components to connect. Apart from that, refinement of LGWF-nets is defined at the level of a complete net rather than specific places and transitions. Refinement preserves the soundness of LGWF-net components and an interface, which describes interactions at the abstract level.

2.6 Conclusions of Chapter 2

This chapter studied a semantically correct composition of interacting workflow nets. We developed an approach to modeling their synchronous and asynchronous interactions using two kinds of transition labels. We defined an *asynchronous-synchronous composition* that may not preserve the soundness of interacting components. To overcome this problem, we use an *interface* model describing how components interact. The interface represents an abstract view of a complete multi-agent system. There is a subnet in the interface representing the behavior of an agent. The correspondence between an interface and agent models is determined using α-morphisms. Structural and behavioral properties of the abstraction/refinement relation based on α-morphisms helped us to prove that refining subnets in the interface with sound models preserves the interface soundness.

We identify two main advantages of the proposed AS-composition. Firstly, the problem of constructing a correct composition of workflow nets is resolved at the

abstract level. Refinement of places in the interface pattern requires checking structural and only local behavioral constraints. Sound models of interface patterns can be reused for different component refinements. Secondly, AS-composition leaves asynchronous channels and synchronous transitions open for others to interact.

The main limitation of the AS-composition lies in defining α-morphisms from component models towards the interface pattern. The following chapter is devoted to overcoming this difficulty.

Chapter 3

Transformations of LGWF-Nets

The preservation of soundness in the AS-composition of labeled GWF-nets (see Theorem 2) is based on defining $\widehat{\alpha}$-morphisms from agent nets towards the corresponding parts in an interface pattern. The direct application of Definition 14 is rather difficult due to the complex global structural constraints imposed on related LGWF-nets. In this chapter, we define structural LGWF-net transformations, which induce an $\widehat{\alpha}$-morphism between an initial and transformed LGWF-net. A collection of *local* transformations, which change only the specific subnets in LGWF-nets, is proposed. Structural transformations are the heart of the second correctness aspect of the compositional process discovery algorithm. In addition, we study the key properties of these transformations.

3.1 Step-Wise Definition of Morphisms

Let $\varphi\colon N_1 \to N_2$ be an $\widehat{\alpha}$-morphism. The main idea of the step-wise definition is to apply a sequence of local transformations to construct N_1 (N_2) from N_2 (N_1), as shown in Fig. 18. Transformations are called *local*, since they affect only some subnet, while the rest of the model remains unchanged. In other words, transformations are applied to an abstraction (refinement) in order to construct

its refinement (abstraction). The application of a transformation implies checking only local structural constraints imposed on the specific subnet.

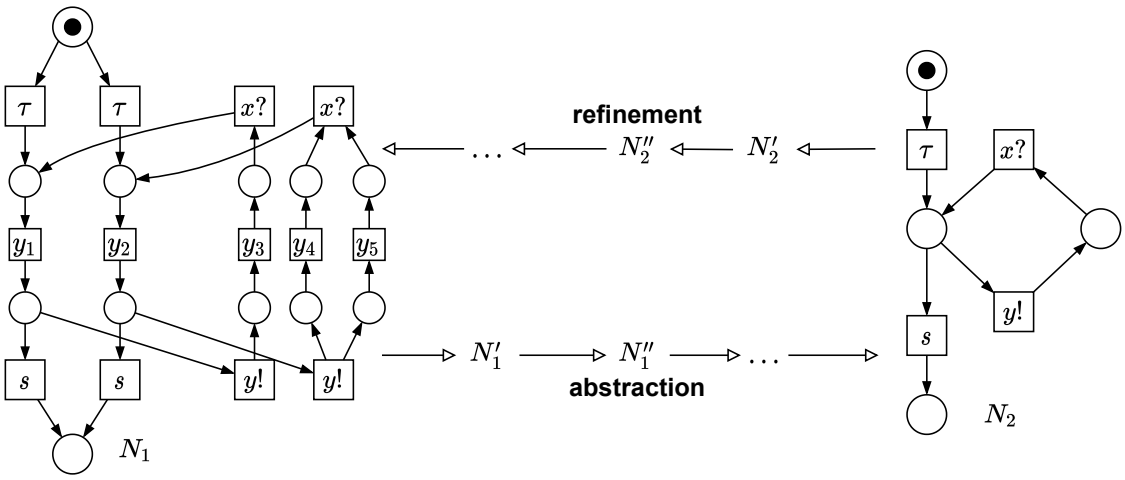


Figure 18: Step-wise definition of morphisms: sequence of transformations

Given an LGWF-net N, one can construct several possible abstractions of N (by Definition 14) depending on the desired detail level. Using transformations helps to reduce this ambiguity. On the one hand, transformations should preserve interacting transitions, since an abstraction of an LGWF-net is supposed to be mapped on the interface pattern. On the other hand, transformations should minimize the number of local transitions in an LGWF-net corresponding to the internal behavior of an agent (a multi-agent system).

However, refinement transformations should only respect interacting transitions. They do not require constraints on minimizing a number of local transitions.

Firstly, while developing transformations of LGWF-nets, we will take into account the requirements imposed by the definition of $\widehat{\alpha}$-morphisms. Secondly, transformations should also respect the structural requirements of LGWF-nets (Definition 7 and Definition 10), i. e., a transformed model should not fall outside the class of LGWF-nets. Thus, the situation when a local transition cannot be reduced is possible since this reduction may violate the structural requirements.

We next define the main components of a transformation *rule* and discuss when a transformation rule can be applied to a given LGWF-net.

For what follows, let $N = (P, T, F, m_0, m_f, h, k)$ be an LGWF-net, where $h\colon T \to \Lambda \cup \{\tau\}$ is a total transition labeling function, and $k\colon P \nrightarrow \mathfrak{C}$ is a partial injective place labeling function, which assigns labels to places connecting transitions with complement asynchronous labels.

We define transformation *rules*, which induce $\widehat{\alpha}$-morphisms. A transformation rule is a tuple $\rho = (L, c_L, R, c_R)$, where:

1. L is the *left* part of a rule that is a subnet in an LGWF-net to be transformed.
2. c_L – flow relation and labeling constraints imposed on L.
3. R is the *right* part of a rule that is a subnet replacing L in an LGWF-net.
4. c_R – flow relation, marking, labeling constraints imposed on R.

L and c_L define the *applicability* constraints of a transformation rule, whereas R and c_R define the transformation itself. We do not give the complete formalization of c_L and c_R since the specific constraints of applicability and transformation are discussed in the rule definitions. These constraints are required to define an $\widehat{\alpha}$-morphism between an initial and transformed LGWF-net. An $\widehat{\alpha}$-morphism induced by a transformation rule ρ is denoted φ_ρ.

Then a transformation rule $\rho = (L, c_L, R, c_R)$ is *applicable* to an LGWF-net N if there exists a subnet in N isomorphic to L with respect to structural and labeling constraints c_L.

Let $\rho = (L, c_L, R, c_R)$ be a transformation rule applicable to N. Let $N(X_L)$ be the subnet of N, generated by $X_L \subseteq P \cup T$, s.t. it is isomorphic to L. Then we equivalently say that ρ is applicable to the subnet $N(X_L)$ in N. The application of ρ to N includes the following steps, as shown in Fig. 19:

1. Remove the subnet $N(X_L)$ from N.

2. Add the subnet corresponding to the right part R of ρ to N connecting it with the nodes in the neighborhood ${}^\bullet X_L {}^\bullet$ of the removed subnet.

3. Make necessary changes, i.e., relabel transitions and add tokens to places, in the inserted subnet according to c_R.

The result of applying ρ to a subnet $N(X_L)$ in N is a new LGWF-net denoted by $\rho(N, X_L) = (P', T', F', m_0', m_f', h', k')$. We use the "functional" notation $\rho(N, X_L)$, explicitly specifying the subnet affected in N to avoid the ambiguity when a transformation rule can be applied to several subnets in an LGWF-net. For brevity, we also write $N \xrightarrow{\rho} N'$ if $N' = \rho(N, X_L)$, and the specification of the transformed subnet is not important or clear from the context.

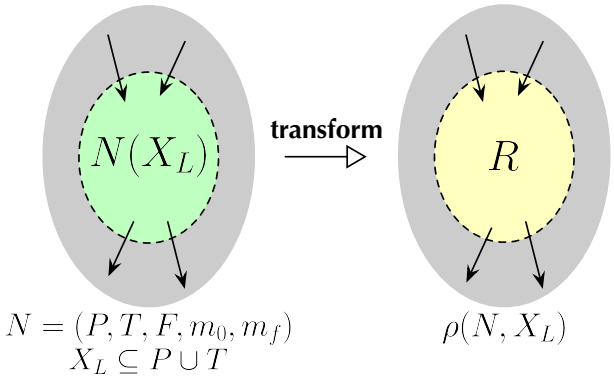


Figure 19: Transformation rule

In the following two sections, *abstraction* and *refinement* rules are discussed. Given an $\widehat{\alpha}$-morphism $\varphi\colon N_1 \to N_2$, abstraction rules are used to construct N_2 from N_1, while refinement rules are used to construct N_1 from N_2. The specifications of a specific abstraction and refinement rule correspond to the key components of a transformation rule discussed above.

3.2 Abstraction Rules

Here, we propose a set of five simple abstraction rules. They are used to abstract LGWF-nets, s. t. there is an $\widehat{\alpha}$-morphism from an initial LGWF-net N towards the result of applying a sequence of abstraction rules to N. We show that abstraction rules induce $\widehat{\alpha}$-morphisms and preserve not only reachable markings, but also deadlocks in LGWF-nets. This is a somewhat stronger property in comparison with the reachable marking preservation stated in Proposition 5.

In the following, let $N = (P, T, F, m_0, m_f, h, k)$ be an LGWF-net, and $N' = \rho(N, X_L) = (P', T', F', m_0', m_f', h', k')$ be the result of applying ρ to N.

Rule A1: Place simplification

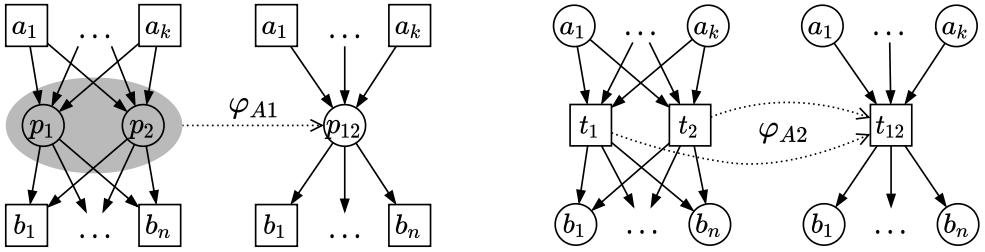
- *applicability constraints*: two unlabeled places $p_1, p_2 \in P$ in N, i. e., $p_1, p_2 \notin \mathrm{dom}(k)$, with the same neighborhood (${}^\bullet p_1 = {}^\bullet p_2$ and $p_1{}^\bullet = p_2{}^\bullet$), as shown in Fig. 20a.
- *transformation:* fusion of p_1 and p_2 into a single unlabeled place p_{12} that preserves the neighborhood of p_1 and p_2, i. e., ${}^\bullet p_{12} = {}^\bullet p_1 = {}^\bullet p_2$, $p_{12}{}^\bullet = p_1{}^\bullet = p_2{}^\bullet$. Also, $p_{12} \notin \mathrm{dom}(k')$ and $p_{12} \in m_0' \Leftrightarrow (p_1 \in m_0$ and $p_2 \in m_0)$.
- *$\widehat{\alpha}$-morphism* $\varphi_{A1}\colon N \to N'$, where $N' = \rho_{A1}(N, \{p_1, p_2\})$, maps places p_1 and p_2 in N to place p_{12} in N'. For the rest of nodes in N, φ_{A1} is the identity mapping between N and N'.

Place simplification is among the most basic Petri net transformations. Places that can be simplified (fused) do not restrict the behavior of a Petri net. It was discussed earlier, for instance, in [43] (cf. "fusion of parallel places") and in [44] (cf. "simplification of redundant places").

Rule A2: Transition simplification

- *applicability constraints*: two transitions $t_1, t_2 \in T$ in N with the same label ($h(t_1) = h(t_2)$) and neighborhood (${}^\bullet t_1 = {}^\bullet t_2$ and $t_1{}^\bullet = t_2{}^\bullet$).

- *transformation:* fusion of t_1 and t_2 into a single transition t_{12} that preserves both the labels and the neighborhood of t_1 and t_2, i. e., $h'(t_{12}) = h(t_1) = h(t_2)$, ${}^\bullet t_{12} = {}^\bullet t_1 = {}^\bullet t_2$, and $t_{12}{}^\bullet = t_1{}^\bullet = t_2{}^\bullet$.
- $\widehat{\alpha}$*-morphism* $\varphi_{A2} \colon N \to N'$, where $N' = \rho_{A2}(N, \{t_1, t_2\})$, maps transitions t_1 and t_2 in N to transition t_{12} in N'. For the rest of nodes in N, φ_{A2} is the identity mapping between N and N'.



(a) abstraction rule A1 (b) abstraction rule A2

Figure 20: Place and transition simplification

Figure 20b shows the left and right parts of the abstraction rule ρ_{A2}. Two transitions t_1 and t_2 can be fused only if they have the same label, as required by Definition 14. In this case, the result of their fusion, transition t_{12}, should inherit the labels of t_1 and t_2.

Transition simplification (without labeling constraints) is one of the basic Petri net transformations as well. It was considered, for instance, in [43] (cf. "fusion of parallel transitions").

Rule A3: Local transition elimination

- *applicability constraints:* a local transition $t \in T$ in N, s. t. $h(t) \notin n$ and:
  1. ${}^\bullet t = \{p_1\}$ and $t^\bullet = \{p_2\}$.
  2. $p_1{}^\bullet = {}^\bullet p_2 = \{t\}$.
  3. ${}^\bullet p_1 \neq \varnothing$ or $p_2{}^\bullet \neq \varnothing$.

4. ${}^\bullet p_1 \cap p_2{}^\bullet = \varnothing$.

- *transformation:* fusion of t, p_1 and p_2 into a single place p_{12} where ${}^\bullet p_{12} = {}^\bullet p_1$, $p_{12}{}^\bullet = p_2{}^\bullet$, and $p_{12} \in m_0' \Leftrightarrow (p_1 \in m_0$ or $p_2 \in m_0)$.
- *$\widehat{\alpha}$-morphism* $\varphi_{A3}\colon N \to N'$, where $N' = \rho_{A3}(N, \{p_1, t, p_2\})$, maps t, p_1, and p_2 in N to place p_{12} in N'. For the rest of nodes in N, φ_{A3} is the identity mapping between N and N'.

Figure 21 shows the left and right parts of the rule ρ_{A3} as well as the construction of the $\widehat{\alpha}$-morphism φ_{A3}. The applicability constraints of ρ_{A3} are aimed to avoid generating isolated places and self-loops in $\rho_{A3}(N, \{p_1, t, p_2\})$, which otherwise will contradict the structural requirements of LGWF-nets.

A similar transition transformation "pre-fusion" was discussed in [44], where it has been expressed as the fusion of two transitions connected by a place.

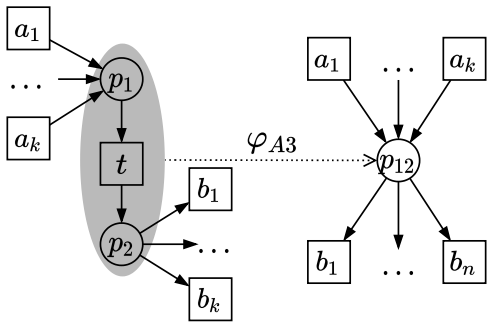


Figure 21: Abstraction rule A3: Local transition elimination

Abstraction rules ρ_{A1}, ρ_{A2} and ρ_{A3} can be generalized: to sets of places and transitions (ρ_{A1} and ρ_{A2}, respectively) or to a "chain" of unlabeled transitions (ρ_{A3}). The example of these generalizations for rules ρ_{A1} and ρ_{A2} is shown in Fig. 22. However, we propose applying a simple abstraction rule multiple times in a row rather than complicating their applicability constraints and left parts, respectively.

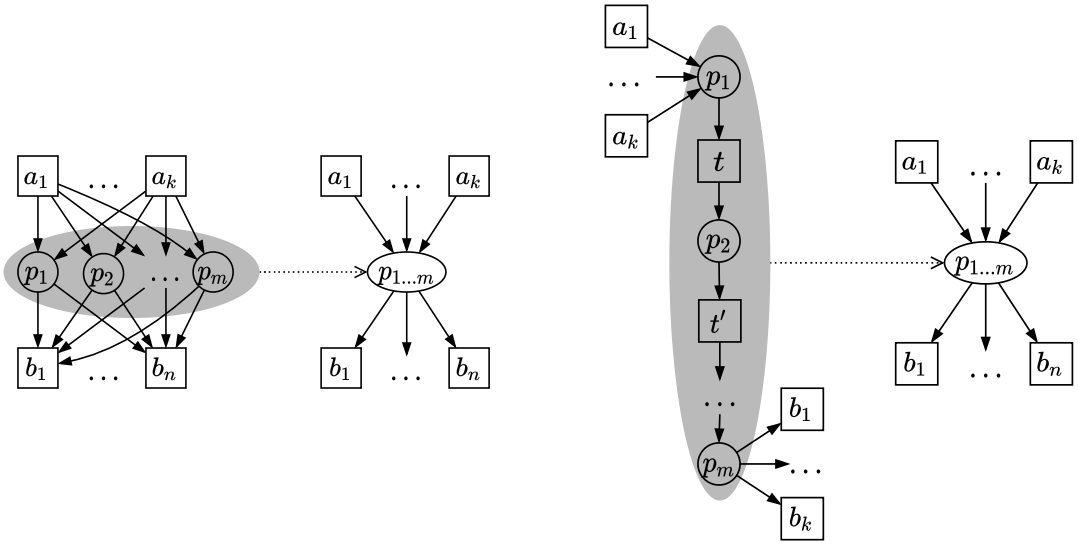


Figure 22: Generalizing abstraction rules ρ_{A1} and ρ_{A3}

Rule A4: Postset-empty place simplification

- *applicability constraints:* two places p_1 and p_2 in N, s. t. ${p_1}^\bullet = {p_2}^\bullet = \varnothing$ and:

  1. ${}^\bullet p_1 \cap {}^\bullet p_2 = \varnothing$.
  2. $\forall C \subseteq P$: if $N(C \cup {}^\bullet C^\bullet)$ is a sequential component, then $p_1 \in C \Leftrightarrow p_2 \in C$.

- *transformation:* fusion of p_1 and p_2 into a single place p_{12}, s. t. ${}^\bullet p_{12} = {}^\bullet p_1 \cup {}^\bullet p_2$, ${p_{12}}^\bullet = {p_1}^\bullet = {p_2}^\bullet$ and $m'_f = (m_f \setminus \{p_1, p_2\}) \cup p_{12}$, as shown in Fig. 23.

- *α-morphism* φ_{A4}: $N \to N'$, where $N' = \rho_{A4}(N, \{p_1, p_2\})$, maps p_1 and p_2 in N to the same place p_{12} in N'. For the rest of nodes in N, φ_{A4} is the identity mapping between N and N'.

Places with the empty postset (in the final marking of N) can be fused only if there is no sequential component that distinguishes p_1 and p_2. This requirement helps us to preserve state machine decomposability in N'. Therefore we also satisfy the requirement 5e of Definition 14.

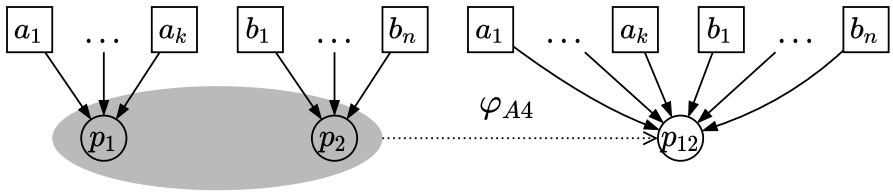


Figure 23: Abstraction rule A4: Postset-empty place simplification

Rule A5: Preset-disjoint transition simplification

In this abstraction rule ρ_{A5}, we fuse two transitions that have the same postset and disjoint presets, as opposed to abstraction rule ρ_{A2}. The applicability constraints of this rule do not allow us to lose deadlocks present in an initial LGWF-net by transforming it. The problem of losing deadlocks is the consequence of the fact that α-morphisms do not *reflect* reachable markings without additional restrictions, as discussed in Section 2.3. In the setting of our study, this means that an inverse image of a reachable marking that enables transitions in an abstract model may be a deadlock in an initial model, as shown in Fig. 24.

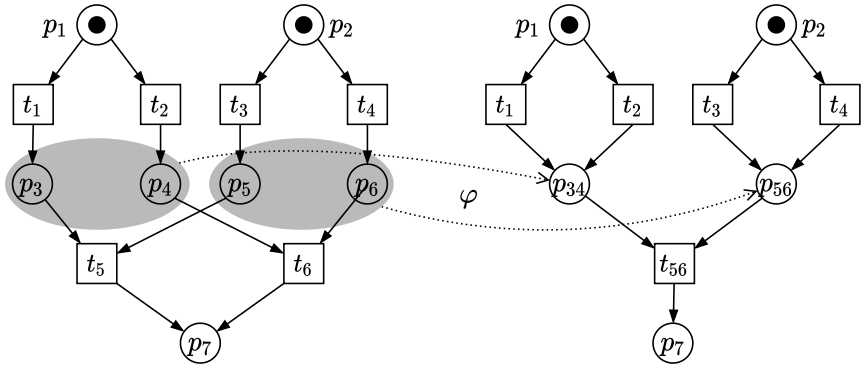


Figure 24: Deadlocks are not preserved by $\widehat{\alpha}$-morphisms

Let us illustrate the problem of losing deadlocks by the following example based on the net system previously shown in Fig. 4. Recall that this net system has two deadlocks $\{p_3, p_6\}$ and $\{p_4, p_5\}$ reachable from the initial marking $\{p_1, p_2\}$. These deadlocks are caused by the fact that conflicts are resolved independently

by two sequential components. Suppose that the two transitions t_5 and t_6 have the same label "c!". Then, using Definition 14, it is possible to fuse p_3 with p_4, p_5 with p_6 and t_5 with t_6 correspondingly (see Fig. 24 where the place and transition fusion is indicated by the indices, and the $\widehat{\alpha}$-morphism φ is shown via dotted arrows). The image t_{56} of t_5 and t_6 has two places in its preset, and there exists reachable marking $\{p_{34}, p_{56}\}$ enabling t_{56}. However, there exists an inverse image of the marking $\{p_{34}, p_{56}\}$, e. g., the deadlock $\{p_3, p_6\}$ that does not enable the inverse image of t_{56}. Correspondingly, the abstraction of a net system with a deadlock may become deadlock-free.

Thus, it is necessary to impose the additional constraints on places in the presets of two transitions to be fused so that if there is a deadlock containing places in the presets of these transitions, then it should not be possible to fuse them. **Preset-disjoint transition simplification** is defined as follows:

- *applicability constraints:* two transitions t_1 and t_2 in N with the same label $h(t_1) = h(t_2)$, s. t.:
  1. ${}^\bullet t_1 \cap {}^\bullet t_2 = \varnothing$ and $|{}^\bullet t_1| = |{}^\bullet t_2|$.
  2. $t_1{}^\bullet = t_2{}^\bullet$.
  3. $\forall a \in {}^\bullet t_1 \, \forall b \in {}^\bullet t_2 \, \exists C \subseteq P\colon a, b \in C$ and $N(C \cup {}^\bullet C^\bullet)$ is a sequential component.
- *transformation:* fusion of t_1 and t_2 into a single transition t_{12} with $h'(t_{12}) = h(t_1) = h(t_2)$, $t_{12}{}^\bullet = t_1{}^\bullet = t_2{}^\bullet$, and ${}^\bullet t_{12} = \{(a, b) \mid a \in {}^\bullet t_1, b \in {}^\bullet t_2, g(a) = b\}$, where $g\colon {}^\bullet t_1 \to {}^\bullet t_2$ is a bijection. The input transitions of ${}^\bullet t_1$ and ${}^\bullet t_2$ are preserved, i. e., $\forall (a, b) \in {}^\bullet t_{12}\colon {}^\bullet(a, b) = {}^\bullet a \cup {}^\bullet b$. As for the initial marking m_0' in $\rho_{A5}(N, \{t_1, t_2\})$, we have $\forall (a, b) \in {}^\bullet t_{12}\colon (a, b) \in m_0' \Leftrightarrow (a \in m_0$ or $b \in m_0)$.
- *α-morphism* $\varphi_{A5}\colon N \to N'$, where $N' = \rho_{A5}(N, \{t_1, t_2\})$, maps transitions t_1 and t_2 to the transition t_{12} in N' as well as every pair of places $a \in {}^\bullet t_1$ and $b \in {}^\bullet t_2$, where $g(a) = b$, is mapped to the place $(a, b) \in {}^\bullet t_{12}$. For other nodes in N, φ_{A5} is the identity mapping.

Fogure 25 provides the left and right parts of the abstraction rule ρ_{A5} where the pairwise fusion of places is shown only for places a_1 and b_1 with $g(a_1) = b_1$ that are fused into the place $f_1 = (a_1, b_1)$. For other pairs of places, this fusion is performed similarly. The bijection $g\colon {}^\bullet t_1 \to {}^\bullet t_2$ is an integral part of ρ_{A5}, which makes the preset-disjoint transition simplification unambiguous.

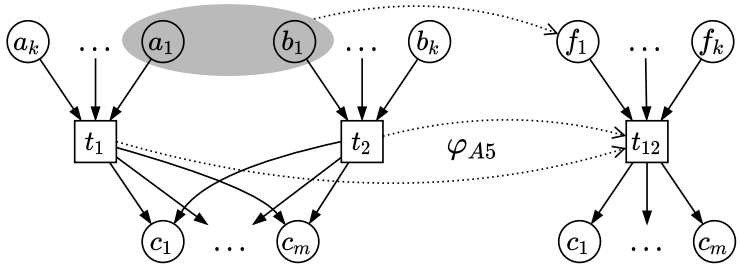


Figure 25: Abstraction rule A5: Preset-disjoint transition simplification

The third applicability constraint of ρ_{A5} makes sure that every place in ${}^\bullet t_1$ is in conflict with every place in ${}^\bullet t_2$. Then it is easy to check that if there is a reachable marking in N with a token in ${}^\bullet t_1$, then there cannot be a token in ${}^\bullet t_2$ at the same time. The application of this rule involves pairwise place fusion in ${}^\bullet t_1$ and ${}^\bullet t_2$. According to the requirement on sequential components, we define a bijection $g : {}^\bullet t_1 \to {}^\bullet t_2$ and fuse places in ${}^\bullet t_1$ and ${}^\bullet t_2$ corresponding by g.

Let us consider two more detailed examples of applying the abstraction rule ρ_{A5}. There are two LGWF-nets N_1 and N_2 shown in Fig. 26. Transitions t_1 and t_2 in N_1 as well as transitions e_1 and e_2 in N_2 are candidates to be fused since they have the same label c!, share the same postset, whereas their presets are disjoint. We have to check whether places in the presets of these transitions are connected by sequential components. The results of this verification for N_1 and N_2 are given in Table 3, where we provide only sets of places corresponding to sequential components.

In N_1, there is no sequential component containing places p_1 and p_4. Indeed, there is the deadlock $\{p_1, p_6, p_4\}$ containing places both from ${}^\bullet t_1$ and ${}^\bullet t_2$. Thus, transitions t_1 and t_2 in N_1 cannot be fused without losing this deadlock.

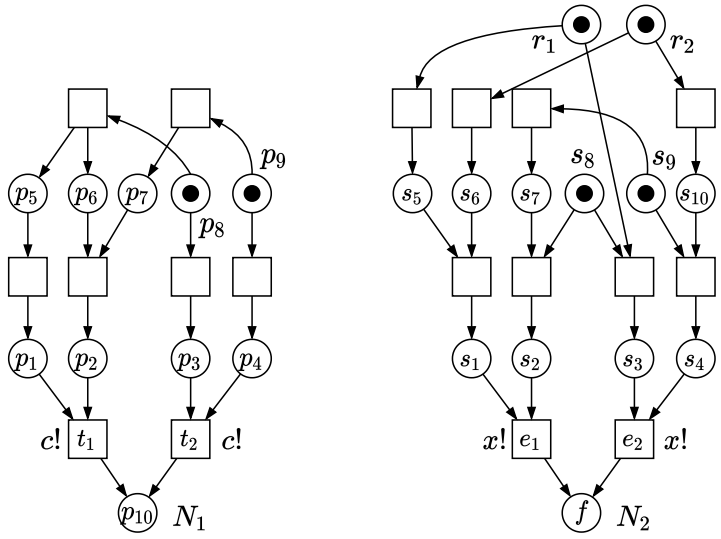
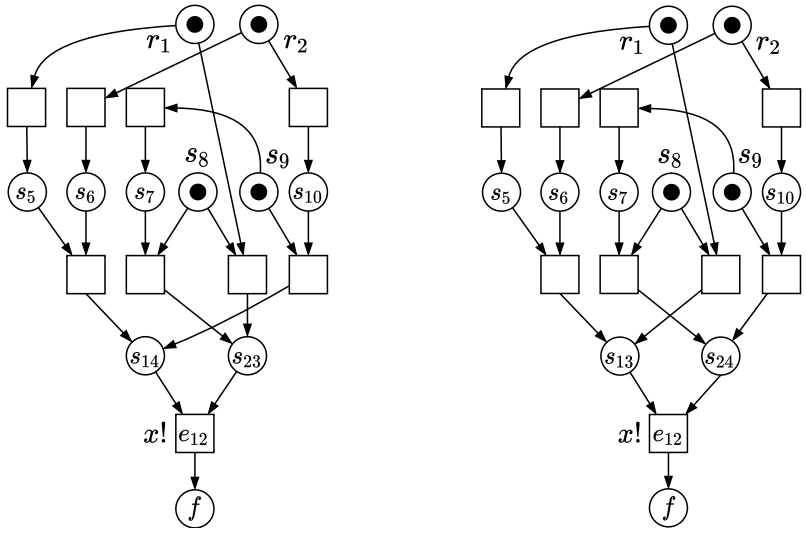


Figure 26: Two LGWF-nets to check the applicability constraints of ρ_{A5}

In N_2, we have found four sequential components for all pairs of places from ${}^\bullet e_1$ and ${}^\bullet e_2$. Thus, we can fuse these transition according to the abstraction rule ρ_{A5}. There can be two possible transformations depending on the choice of places to be fused, i. e., either s_1 is fused with s_4 (see Fig. 27a) or s_1 is fused with s_3 (see Fig. 27b). It is enough to choose a single pair of places to be fused, and the other pair of places is determined in the only possible way.

Table 3: Verification of sequential components in N_1 and N_2 from Fig. 26

Sequential components in N_1		Sequential components in N_2	
p_1 and p_3	$\{p_8, p_5, p_3, p_1, p_{10}\}$	s_1 and s_3	$\{r_1, s_5, s_1, s_3, f\}$
p_1 and p_4	NO	s_1 and s_4	$\{r_2, s_6, s_{10}, s_1, s_4, s_{13}\}$
p_2 and p_3	$\{p_8, p_6, p_2, p_3, p_{10}\}$	s_2 and s_3	$\{s_8, s_2, s_3, f\}$
p_2 and p_4	$\{p_9, p_7, p_2, p_4, p_{10}\}$	s_2 and s_4	$\{s_9, s_7, s_2, s_2, f\}$



(a) s_1 is merged with s_4 (b) s_1 is merged with s_3

Figure 27: Two results of applying the rule ρ_{A5} to N_2 from Fig. 26

3.3 Properties of Abstraction Rules

Here, we discuss the main properties of the simple abstraction rules. We denote the set of abstraction rules by $\mathcal{AR} = \{\rho_{A1}, ..., \rho_{A5}\}$.

By construction, applying an abstraction rule induces an α-morphism from the initial LGWF-net towards the transformed one.

Proposition 8: Abstraction rules induce $\widehat{\alpha}$-morphisms

Let $\rho \in \mathcal{AR}$, s. t. ρ is applicable to a subnet $N(X_L)$ in N. Then there is an $\widehat{\alpha}$-morphism $\varphi_\rho \colon N \to \rho(N, X_L)$.

Corollary 2: Sequences of abstraction rules induce $\widehat{\alpha}$-morphisms

Let $\rho_1, \rho_2 \in \mathcal{AR}$, s.t. ρ_2 is applicable to a subnet in $\rho_1(N, X_L)$ generated by X'_L. Then there is an α-morphism $\varphi_{\rho_2} \circ \varphi_{\rho_1} \colon N \to \rho_2(\rho_1(N, X_L), X'_L)$.

Thus, it is possible to redefine the notion of abstraction without referring to the definition of $\widehat{\alpha}$-morphisms.

Definition 15: Abstraction of LGWF-net

Let N_1 and N_2 be two LGWF-nets. N_1 is an abstraction of N_2 if there exists a sequence of abstraction rules $\pi = \langle \rho_1 \rho_2 ... \rho_n \rangle \in \mathcal{AR}^*$, which leads from N_2 to N_1, i.e., $N_2 \xrightarrow{\rho_1} N_2' \xrightarrow{\rho_2} ... \xrightarrow{\rho_n} N_1$

The important property is whether the order of applying abstraction rules matters when at least two abstraction rules are applicable to the same LGWF-net. In this case, we distinguish whether these abstraction rules coincide or differ.

Proposition 9: When the order of abstraction rules does not matter

Let $\rho_1, \rho_2 \in \mathcal{AR}$, s.t. ρ_1 is applicable to a subnet $N(X_L^1)$ in N, ρ_2 is applicable to a subnet $N(X_L^2)$ in N and $X_L^1 \neq X_L^2$. Then:

1. If $\rho_1 = \rho_2$, then the effect of applying ρ_2 to $\rho_1(N, X_L^1)$ is isomorphic to the effect of applying ρ_1 to $\rho_2(N, X_L^2)$.
2. If $\rho_1 \neq \rho_2$ and $X_L^1 \cap X_L^2 = \varnothing$, then $\rho_2(\rho_1(N, X_L^1), X_L^2) = \rho_1(\rho_2(N, X_L^2), X_L^1)$.

The second part of Proposition 9 is easy to verify, i. e., the order of applying abstraction rules transforming disjoint subnets is immaterial. However, the first part of this Proposition requires an additional clarification, when $\rho_1 = \rho_2 = \rho_{A5}$. The result of applying the other abstraction rules fully depends on the subnets corresponding to their left parts, which are fixed in Proposition 9.

As discussed above, the bijection g between the input places of two transitions is the integral part of ρ_{A5}. Then we require that for the repeated application of ρ_{A5}, one fixes the bijections at the time of checking the applicability constraints. The following example (see Fig. 28) shows a case when ρ_{A5} can be applied twice and explains how to define the correct bijections between input places.

Suppose an LGWF-net N has a subnet shown in Fig. 28a, which satisfies the applicability constraints of ρ_{A5} since transitions t_1, t_2 and t_3 have the same label. We need to define two bijections between the input places of any two pairs of

transitions, and the bijection for the third pair of transitions will be obtained transitively. For instance, let $g_1 \colon {}^\bullet t_1 \to {}^\bullet t_2$ and $g_2 \colon {}^\bullet t_1 \to {}^\bullet t_3$, s. t. $g_1(a_1) = b_1$, $g_1(a_2) = b_2$, $g_2(a_1) = c_2$, and $g_2(a_2) = c_1$. These correspondences between the input places are also shown by dotted lines in Fig. 28a. Then the third bijection $g_3 \colon {}^\bullet t_2 \to {}^\bullet t_3$ is defined as follows: $g_3(b_2) = c_1$ and $g_3(b_1) = c_2$. Arbitrary definition of the third bijection might break transitivity and, thus, disable the repeated application of the abstraction rule ρ_{A5}. In other words, the number of required bijections corresponds to the number of times ρ_{A5} will be applied to N.

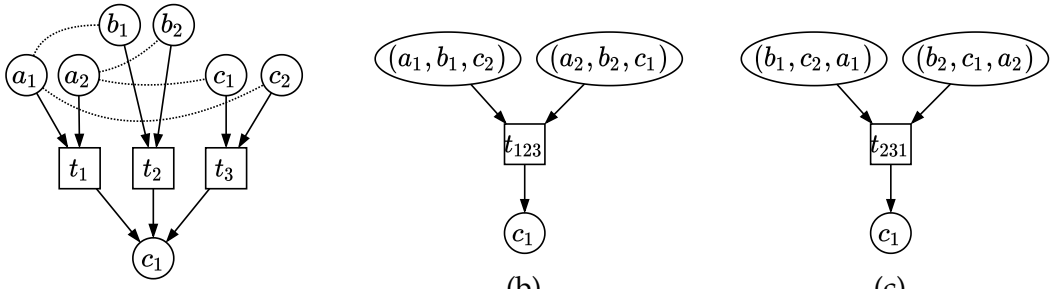


Figure 28: Repeated application of the abstraction rule ρ_{A5}

We next demonstrate that the order of fusing transitions is not important since the results are isomorphic. Suppose that, firstly, transitions t_1 and t_2 are to be fused. Then they are transformed into a single transition t_{12}, s. t. ${}^\bullet t_{12} = \{(a_1, b_1), (a_2, b_2)\}$ according to g_1. The fusion of t_{12} with t_3 will yield a transition t_{123} with ${}^\bullet t_{123} = \{(a_1, b_1, c_2), (a_2, b_2, c_1)\}$, as shown in Fig. 28b. Changing the order of the consecutive fusions, we may, for example, obtain a transition t_{231} with ${}^\bullet t_{231} = \{(b_2, c_1, a_2), (b_1, c_2, a_1)\}$ (see Fig. 28c), which is isomorphic to the earlier constructed result.

The unambiguity of the repeated application of ρ_{A5} requires that the bijections between the input places of transition pairs are defined for an initial LGWF-net.

According to the structural requirements of abstraction rules AR, we also conclude that if there is a deadlock in an initial LGWF-net, then the image of this

deadlock is also a deadlock in a transformed LGWF-net (see Proposition 10). In proving this statement, we rely on the fact that α-morphisms *preserve* reachable markings and transition firings (see Proposition 5), i. e., an image of a reachable marking in a refined LGWF-net is also a reachable marking which, moreover, enables any image of enabled transitions in a refined model.

Proposition 10: Abstraction rules preserve deadlocks

Let $N = (P, T, F, m_0, m_f, h, k$ be an LGWF-net. Let $\rho \in AR$, s. t. ρ is applicable to a subnet $N(X_L)$ in N. Let $m \in [m_0\rangle$ be a deadlock in N. Then $\varphi_\rho(m)$ is a deadlock in $\rho(N, X_L)$.

Proof. Let $N' = \rho(N, X_L)$. If $m^\bullet = \varnothing$, then, by Definition 14, $\varphi_\rho(m)^\bullet = \varnothing$. Thus, $\varphi_\rho(m)$ is a deadlock in N'. If $\exists t \in T$: ${}^\bullet t \cap m \neq \varnothing$, then the proof is done by contradiction. Suppose that $\varphi_\rho(m)$ is not a deadlock. Then either ${}^\bullet\varphi_\rho(t) = \varphi_\rho(m)$, i.e., a transition t and ${}^\bullet t^\bullet$ is mapped to the same place, or ${}^\bullet\varphi_\rho(t) \subseteq \varphi_\rho(m)$, i.e., a marking $\varphi_\rho(m)$ enables $\varphi_\rho(t)$ in N'. A transition t cannot be mapped to a place by φ_ρ since $|{}^\bullet t| > 1$, because there are places in ${}^\bullet t$, s.t. ${}^\bullet t \cap m \neq \varnothing$ and there is at least one place $p \in {}^\bullet t$, s.t. $p \notin m$. If a marking $\varphi_\rho(m)$ enables $\varphi_\rho(t)$ in N', then t is fused with another transition t' by ρ, s.t. ${}^\bullet t' \cap m \neq \varnothing$. This fusion is not allowed by the abstraction rule ρ_{A5}, then there is a contradiction. □

We next consider an example of abstracting an LGWF-net using the sequence above described abstraction rules. Consider the behavior of Agent 1 in the LGWF-net shown earlier in Fig. 1. Figure 26 shows its abstraction, LGWF-net A_1, which is obtained by applying a sequence consisting of 13 abstraction rules (through 6 steps). According to Proposition 9, the application of abstraction rules transforming disjoint subnets is shown as a single step. The corresponding $\widehat{\alpha}$-morphism from the initial LGWF-net towards A_1 is a composition of all intermediate $\widehat{\alpha}$-morphisms induced by concrete abstraction rules. A similar construction can be also done for the behavior of Agent 2 from Fig. 1.

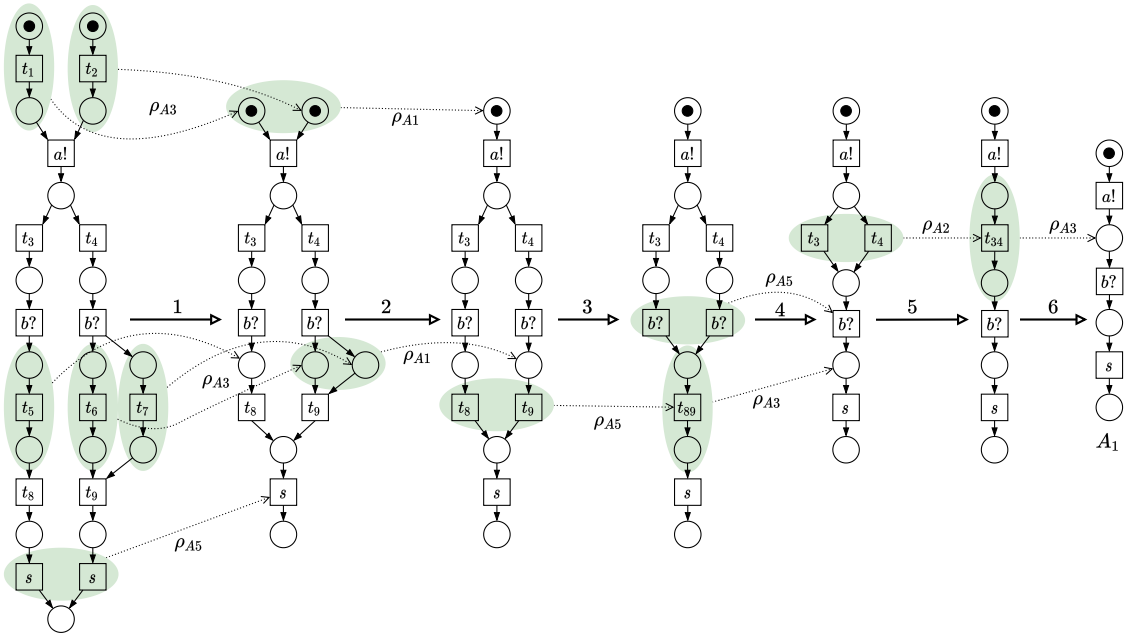


Figure 29: Abstracting the behavior of Agent 1 from Fig. 1

Note also that LGWF-net A_1, shown in Fig. 29 cannot be abstracted further using abstraction rules A1–A5, since it has only three interacting transitions $a!$, $b?$, and s. The characterization of irreducible models is subject for the future research on abstraction transformations.

3.4 Refinement Rules

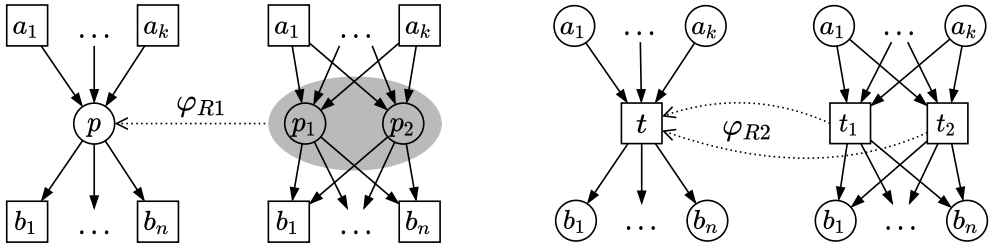
In this section, we define four simple refinement rules. They are used to construct a refinement of a given LGWF-net. Three of four proposed refinement rules are the exact inverse of the abstraction rules presented in Section 3.2. Refinement rules also induce $\widehat{\alpha}$-morphisms. The main difference here is that the direction of $\widehat{\alpha}$-morphisms is opposite to the direction of transformations, i.e., from the transformed LGWF-net towards an initial one.

In the following, let $N = (P, T, F, m_0, m_f, h, k)$ be an LGWF-net. Recall also

that the effect of applying a transformation rule ρ to N is denoted by $\rho(N, X_L) = (P', T', F', m_0', m_f', h', k')$, where $X_L \subseteq P \cup T$ and $N(X_L)$ is a subnet in N, which is transformed by ρ.

Rule R1: Place duplication

- *applicability constraints:* an unlabeled place p in N, i. e., $p \notin \mathrm{dom}(k)$.
- *transformation:* split p into two unlabeled places p_1 and p_2 where ${}^\bullet p_1 = {}^\bullet p_2 = {}^\bullet p$, $p_1{}^\bullet = p_2{}^\bullet = p^\bullet$, ($p_1 \in m_0'$ and $p_2 \in m_0'$) $\Leftrightarrow p \in m_0$, and $p_1, p_2 \notin \mathrm{dom}(k')$, as shown in Fig. 30a.
- *α-morphism* $\varphi_{R1}\colon N' \to N$, where $N' = \rho_{R1}(N, \{p\})$, maps places p_1 and p_2 in N' to place p in N. For the rest of nodes in N', φ_{R1} is the identity mapping between N' and N.



(a) refinement rule R1 (b) refinement rule R2

Figure 30: Place and transition duplication

R2: Transition duplication

- *applicability constraints:* a transition t in N.
- *transformation:* split t into two transitions t_1 and t_2 where ${}^\bullet t_1 = {}^\bullet t_2 = {}^\bullet t$, $t_1{}^\bullet = t_2{}^\bullet = t^\bullet$ and $h'(t_1) = h'(t_2) = h(t)$.
- *α-morphism* $\varphi_{R2}\colon N' \to N$, where $N' = \rho_{R2}(N, \{t\})$, maps transitions t_1 and t_2 in N' to transition t in N. For the rest of nodes in N', φ_{R2} is the identity mapping between N' and N.

Figure 30b shows the left and right parts of the abstraction rule ρ_{R2}. Duplicated transitions t_1 and t_2 in $\rho_{R2}(N, \{t\})$ preserve the label of t.

Rule R3: Local transition introduction

- *applicability constraints:* an unlabeled place p in N, i. e., $p \notin \mathrm{dom}(k)$.
- *transformation:* replacement of p with a local transition t, i. e., $h'(t) \notin In$, and two places p_1, p_2 (see Fig. 31) where:
  1. ${}^\bullet t = \{p_1\}$ and $t^\bullet = \{p_2\}$.
  2. ${p_1}^\bullet = {}^\bullet p_2 = \{t\}$.
  3. ${}^\bullet p_1 = {}^\bullet p$ and ${p_2}^\bullet = p^\bullet$.
  4. $p \in m_0 \Leftrightarrow (p_1 \in m'_0$ and $p_2 \notin m'_0)$.
- *α-morphism* $\varphi_{R3}\colon N' \to N$, where $N' = \rho_3(N, \{p\})$, maps t, p_1 and p_2 in N' to place p in N. For the rest of nodes in N', φ_{R3} is the identity mapping between N' and N.

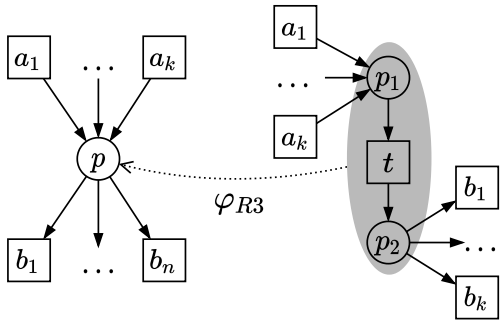


Figure 31: Refinement rule R3: local transition introduction

Refinement rule ρ_{R1} (ρ_{R2}) can be generalized to the case when a place (a transition) in the initial EN system is split into a set of places (transitions). Refinement rule ρ_{R3} can be generalized to the case when a places in the in initial EN system is replaced with a "chain" of local transitions. These extensions are similar to the

possible generalizations of abstraction rules ρ_{A1}, ρ_{A2} and ρ_{A3} discussed above, as shown in Fig. 22.

Rule R4: Place split

- *applicability constraints:* an unlabeled place p in N, i. e., $p \notin dom(k)$ with $|{}^\bullet p| > 1$.
- *transformation:* split p into two unlabeled places p_1 and p_2 (see Fig. 32):
  1. ${}^\bullet p_1 \neq \varnothing$ and ${}^\bullet p_2 \neq \varnothing$.
  2. ${}^\bullet p_1 \subset {}^\bullet p$ and ${}^\bullet p_2 \subset {}^\bullet p$.
  3. ${}^\bullet p_1 \cap {}^\bullet p_2 = \varnothing$ and ${}^\bullet p_1 \cup {}^\bullet p_2 = {}^\bullet p$.
  4. $|p_1{}^\bullet| = |p_2{}^\bullet| = |p^\bullet|$, and there is a bijection $f_i \colon p_i{}^\bullet \to p^\bullet$ with $i = 1, 2$, s.t. for all $t' \in p_i{}^\bullet$, $h'(t') = h(f_i(t'))$.
  5. $(p_i{}^\bullet)^\bullet = (p^\bullet)^\bullet$ with $i = 1, 2$.
  6. ${}^\bullet(p_i{}^\bullet) \setminus \{p_i\} = {}^\bullet(p^\bullet) \setminus \{p\}$ with $i = 1, 2$.
  7. if $p \in m_0$, then $p_1 \in m_0' \Leftrightarrow p_2 \notin m_0'$.
- *α-morphism* $\varphi_{R4} \colon N' \to N$, where $N' = \rho_{R4}(N, \{p\})$, maps places p_1 and p_2 in N' to place p in N and maps each transition $t' \in p_i{}^\bullet$ in N' to a transition $t \in p^\bullet$ in N if $f_i(t') = t$ with $i = 1, 2$. For the rest of nodes in N', φ_{R4} is the identity mapping between N' and N.

While splitting a place p in N, its neighborhood is also split between p_1 and p_2 in $\rho_{R4}(N, \{p\})$. According to constraints 1, 2 and 3, the preset of p is divided into two disjoint, proper and non-empty subsets. According to constraint 4, the postsets of p_1 and p_2 are exactly two copies of the postset of p, s.t. the labels of corresponding transitions are preserved. Moreover, by constraints 5 and 6, the input and output places in $p_1{}^\bullet$ and $p_2{}^\bullet$ are the same as the input and output places of $p^\bullet$. These requirements on splitting the neighborhood of p in N are based on the requirements 5b and 5c of Definition 14.

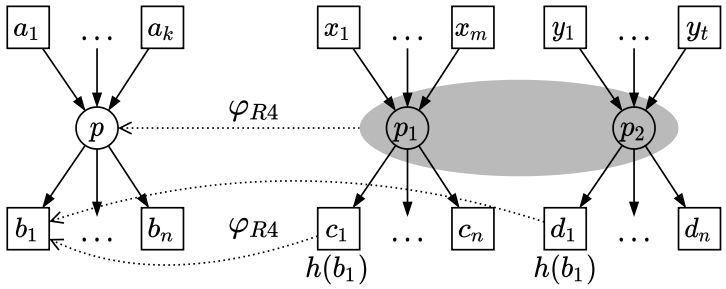


Figure 32: Refinement rule R4: place split

Figure 32 provides the left and right parts of the refinement rule ρ_{R4}, where the corresponding $\widehat{\alpha}$-morphism maps p_1 and p_2 in the transformed LGWF-net N' to a place p in the initial LGWF-net N. The map from the postsets of p_1 and p_2 to the postset of p is shown only for two pairs of transitions: c_1 is mapped to b_1 since $f_1(c_1) = b_1$; d_1 is also mapped to b_1 since $f_2(d_1) = b_1$, where the bijections f_1 and f_2 are defined according to constraint 4 of this refinement rule.

3.5 Properties of Refinement Rules

Here, we discuss the main properties of the proposed refinement rules. Let $RR = \{\rho_{R1}, \ldots, \rho_{R4}\}$ be the set of refinement rules

By construction, the application of a refinement rule induces an $\widehat{\alpha}$-morphism from a transformed LGWF-net to an initial LGWF-net. This also follows from the fact that rules ρ_{R1}, ρ_{R2} and ρ_{R3} are inverse to the abstraction rules ρ_{A1}, ρ_{A2} and ρ_{A3}, respectively.

Proposition 11: Refinement rules induce $\widehat{\alpha}$-morphisms

Let $\rho \in RR$, s. t. ρ is applicable to a subnet $N(X_L)$ in N. Then there is an α-morphism $\varphi_\rho \colon \rho(N, X_L) \to N$.

Corollary 3: Sequences of refinement rules induce $\widehat{\alpha}$-morphisms

Let $\rho_1, \rho_2 \in RR$, s. t. ρ_2 is applicable to a subnet in $\rho_1(N, X_L)$ generated by X'_L. Then there is an α-morphism $\varphi_{\rho_2} \circ \varphi_{\rho_1} \colon \rho_2(\rho_1(N, X_L), X'_L) \to N$.

Thus, similarly to Definition 15, we can redefine the notion of refinement using only refinement transformations.

Definition 16: Refinement of LGWF-net

Let N_1 and N_2 be two LGWF-nets. N_1 is a refinement of N_2 iff there exists a sequence of refinement transformations $\pi = \langle \rho_1 \rho_2 ... \rho_k \rangle \in RR^*$, which leads from N_2 to N_1, i. e., $N_2 \stackrel{\rho_1}{\to} N'_2 \stackrel{\rho_2}{\to} ... \stackrel{\rho_n}{\to} N_1$.

Similar to the abstraction rules, we also observe that application of the refinement rules does not introduce "new" deadlocks to transformed models, i. e., an inverse image of a deadlock in a refined LGWF-net is also a deadlock already present in an initial LGWF-net.

Proposition 12: Refinement rules reflect deadlocks

Let $N = (P, T, F, m_0, m_f, h, k)$ be an LGWF-net. Let $\rho \in RR$, s. t. ρ is applicable to a subnet $N(X_L)$ in LGWF-net N. Let $m' \in [m'_0\rangle$ be a deadlock in $\rho(N, X_L)$. Then $\varphi_\rho(m')$ is a deadlock in N.

Proof. The proof follows from two facts. Firstly, as discussed in Chapter 1, a deadlock m in a net system covered by sequential components $N = (P, T, F, m_0)$ is such a reachable marking, where for any transition $t \in T$, s.t. ${}^\bullet t \cap m \neq \varnothing$, there is at least one place $p \in {}^\bullet t$, s.t. $p \notin m$. Secondly, the application of the refinement rules, which result in splitting places (thus, generating new inverse images (in N') of reachable markings in an initial net system N), fully preserves their neighborhoods. □

The immediate corollary of Proposition 12 is that a refinement of a sound

LGWF-net, obtained via a sequence of transformations, is sound as well.

Corollary 4: Refinement rules preserve soundness

Let N_1 and N_2 be two LGWF-nets, s. t. N_2 is sound. Let N_1 be a refinement of N_2. Then N_1 is a sound LGWF-net.

Therefore an $\widehat{\alpha}$-morphism $\varphi\colon N_1 \to N_2$, where N_1 is a refinement of N_2 in the sense of Definition 16, *reflects* the soundness of N_2.

We next discuss an example symmetrical to the one considered above in Fig. 29. We demonstrate that the behavior of Agent 1 in the LGWF-net shown in Fig. 1 is a refinement of A_1 obtained in Fig. 29. The sequence of refinement rules leading from A_1 to the LGWF-net corresponding to the behavior of Agent 1 is provided in Fig. 33, where transformations affecting disjoint subnets are also shown via a single step. The total sequence includes 13 refinement rules (through 6 steps).

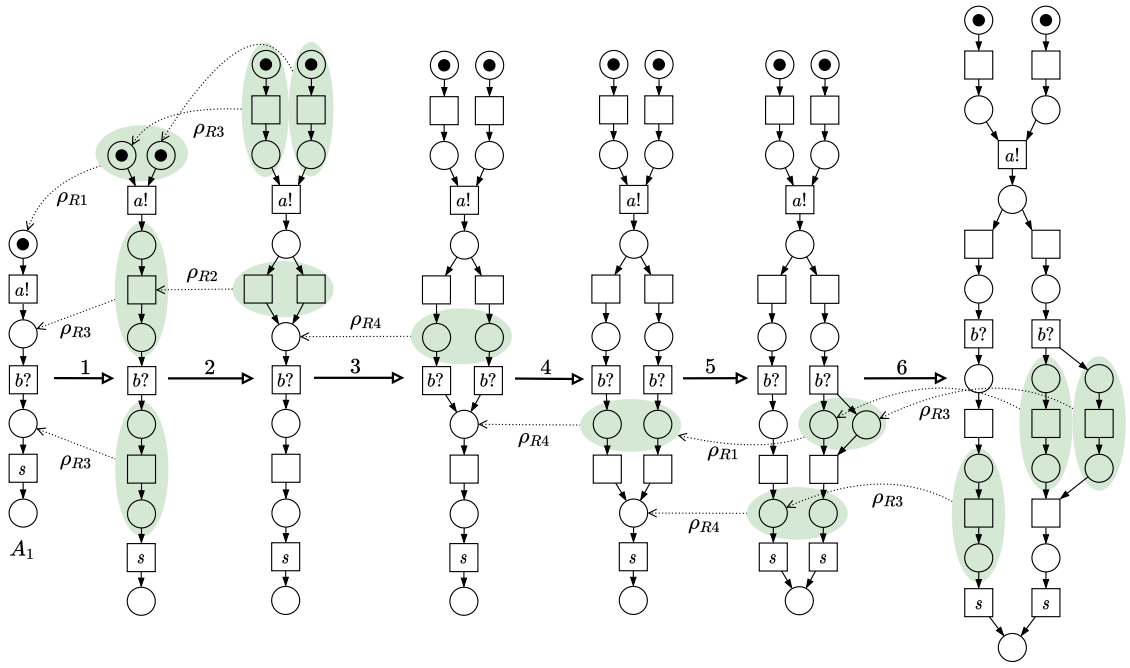


Figure 33: The behavior of Agent 1 shown in Fig. 1 is a refinement of A_1 obtained in Fig. 29

3.6 Related Works: Petri Net Transformations

The construction of the reachability graph of a Petri net faces the well-known *state-explosion* problem when the number of reachable states grows exponentially compared to the size of a Petri net. That is why various *structural* techniques were developed in Petri net theory. Their main advantage is the possibility to verify behavioral properties of Petri nets without computing their reachable markings.

Structural Petri net transformations that preserve classical properties of Petri nets, including *boundedness*, *liveness*, covering by *place invariants* make the verification of concurrent systems easier. Petri net transformations were first introduced in several works [43–47] where the authors defined different families of local reduction and extension rules.

T. Murata [43] discussed a collection of simple reduction/refinement transformations. They preserve liveness, safeness and boundedness of Petri nets. Reduction rules can be used to transform a large system into a smaller one to facilitate the behavioral analysis. Refinement rules are used to synthesize a refined model from an abstract net in a hierarchical manner. We also showed that some abstract transformations, defined in Section 3.2, correspond to the reduction rules from [43], namely place/transition simplification and local transition elimination.

G. Berthelot and G. Roucairol [45] encoded Petri nets via special grammatical representations with production rules corresponding to transition firings. Corresponding Petri net transformations are based on transforming a set of production rules. The important property of these transformations, proven in [45], is the Church-Rosser property, i. e., the reduction process is finite and the order of applying reduction transformations is immaterial. As for the our collection of abstraction transformations A1–A5, inducing $\widehat{\alpha}$-morphisms between LGWF-nets, we proved that the order of applying transformations is irrelevant under the correct choice of the bijection between places in the rule A5 (see Proposition 9).

Petri net transformations, defined in the earlier work [44] by G. Berthelot, include, apart from the variations of place and transition fusions, those introducing

an additional subnet into an original Petri net. A thorough analysis of the property preservation under these transformation was conducted. This paper also studied the important question on the completeness of a transformation system. In other words, the question is whether it is possible to generate all Petri nets from a give one demonstrating the same behavioral properties. This problem was studied from a different point of view: it has been shown that every Petri net in a certain class can be transformed into another Petri net, which is further irreducible.

A step-wise approach to the abstraction and refinement of Petri nets was discussed by R. Valette in [47] and by T. Murata and I. Suzuki in [46]. The authors have considered the substitution of a place (a transition) with a subnet that has two unique input and output transitions. Correspondingly, an input transition should not have input places, while an output transition should not have output places. In addition a subnet replacing a node in a Petri net can also be a well-formed block, which is a live Petri net. Apart from studying the conservation of standard Petri net properties like liveness, boundedness, and safeness, the authors studied the decidability issues connected with these properties.

Free choice Petri nets [48] are also widely adopted to model the behavior of concurrent systems for their structural constraints on conflicts used to express many behavioral properties through structural ones [49]. Intuitively, a conflict in a free choice Petri net is not influenced by the previous transition firings. The work [50] by J. Esparza and M. Silva given a *complete* set of reduction and synthesis transformations supporting top-down modeling of concurrent systems. It is shown that they allow to construct every live and bounded free choice Petri net. By analogy, within the framework of bipolar synchronization schemes [51], studied by H. Genrich P. Thiagarajan, expressible via free choice nets, the authors have also defined reduction and synthesis transformations that yield only correctly behaving synchronization schemes.

Another series of works, including the one by J. Padberg and M. Urbášek [34], and the one by H. Ehrig at al. [52], is devoted to the application of graph trans-

formations in the categorical setting using the double-pushout graph rewriting approach, illustrated by the commutative diagram shown in Fig. 34. These transformations are applied to model and analyze the behavior of re-configurable systems, where different parts of systems can be dynamically replaced and modified. Graph rewriting rules include the left part L and the right part R sharing some nodes in K. Then L (embedded in N_1) is replaced with R (embedded in N_2 after transformation). Common nodes of N_1 and N_2 are in C.

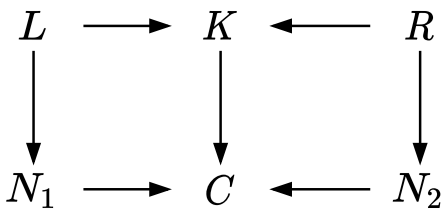


Figure 34: Double-pushout graph rewriting

Place [53] and, more generally, resource (sub-marking) [54] bisimulations, studied by P. Schnoebelen and N. Sidorova and by I. Lomazova, respectively, are other powerful tools that can be used for the reduction of Petri net graphs preserving their observable behavior. These techniques are based on reducing places and resources in Petri nets provided that they exhibit bisimilar behavior. One resource can be replaced by another one if, for instance, it is not accessible in the system. Figure 35, taken from [54], shows the example of reducing a Petri net with weighted arcs based on bisimilar resources. Places p_4 and p_1 can be reduced since they are equivalent to the empty resource not producing any behavior. Place p_5 is reduced, as two tokens in p_5 are equivalent to a single token in p_2 in terms of the behavior.

Morphisms on Petri nets, discussed in Section 2.5 as well, provide a natural framework for the formalization of structural property-preserving relations. Among the others, in [55], the authors have discussed general classes of morphisms on labeled Petri nets inducing various kinds of bisimulations. The class of α-morphisms [15], used in the thesis to formalize mappings of agent behavior on interface patterns, also induces bisimulation between related Petri nets.

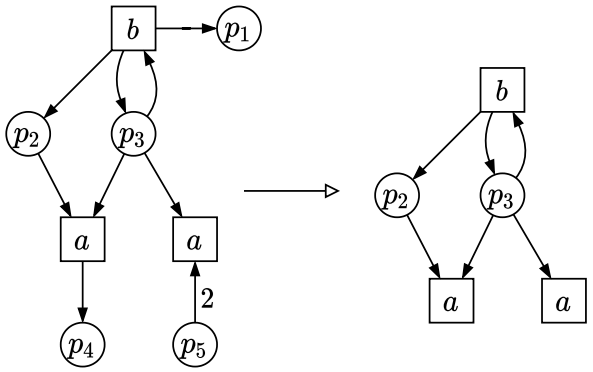


Figure 35: Reduction of a Petri net based on equivalent resources

3.7 Conclusions of Chapter 3

We developed a collection of abstraction/refinement transformations that facilitate the step-wise definition of $\widehat{\alpha}$-morphisms — a restriction of α-morphisms to LGWF-nets, s. t. deadlocks are preserved. Based on the related research discussed above, there are several open questions to be considered in the future:

1. Research on more liberal ways of introducing (for refinement) and detecting (for abstraction) concurrency in nets, as discussed in the following example.
2. Research on the completeness of the abstraction/refinement transformation with respect to property preservation.
3. Characterization of irreducible nets, which cannot be abstracted further.

Consider an $\widehat{\alpha}$-morphism $\varphi\colon N_1 \to N_2$ shown in Fig. 36 where transitions t_3, t_4 and t_5 in N_1 are local. N_2 cannot be obtained form N_1 (and vice versa) by applying the proposed abstraction/refinement rules. For instance, to apply the rule ρ_{A5}, the place p_2 has to be duplicated (the rule ρ_{R1}). However, even in this case, after fusing transitions t_1 and t_2 we will not be able to do pairwise simplification of transitions t_3, t_4 and t_5 since they do not have coincident presets and postsets.

This example demonstrates the need for other ways of introducing and detecting concurrency in LGWF-nets apart from the straightforward duplication of

places. One of the possible directions here is to define more general transformations based on *implicit* places, which do not restrict the behavior of a net system.

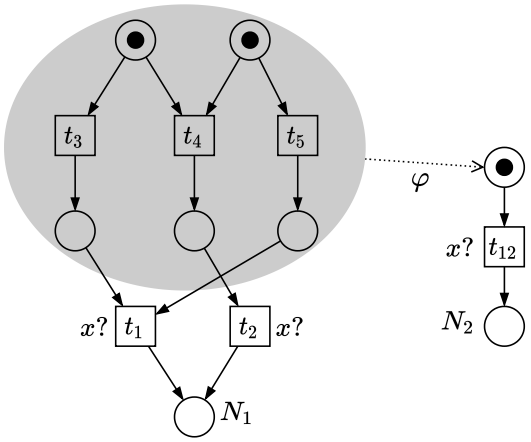


Figure 36: An $\widehat{\alpha}$-morphism that cannot be obtained by transformations

Chapter 4

Compositional Process Discovery

This chapter presents the main algorithm of the compositional discovery designed to synthesize architecture-aware and sound labeled generalized workflow nets from event logs of multi-agent systems. The correctness of this algorithm is formally demonstrated from two perspectives:

1. A process model of a multi-agent system discovered by this algorithm can execute all traces (perfectly fits) an event log.
2. A process model of a multi-agent system discovered by this algorithm is a sound LGWF-net.

We also discuss the third correctness aspect of the compositional process discovery algorithm — a collection of interface patterns modeling typical asynchronous and synchronous interactions among agents.

4.1 The Main Algorithm

The compositional process discovery algorithm (see Algorithm 1) reflects the mains steps of the general scheme of the approach, presented earlier in Fig. 3.

1. DISCOVER(L_{Λ_i}) corresponds to the application of the process discovery algorithm to agent sub-logs. It is important to obtain sound LGWF-nets at this step. For instance, the Inductive miner [6] guarantees the soundness of discovered models.
2. ISREFINEMENT(R_i, A_i) checks if the agent LGWF-net, R_i, is a proper refinement of the corresponding part, A_i, in the interface pattern (by Definition 16, respectively).
3. REPLACE(S, A_i, R_i) substitutes the corresponding part, A_i, in the interface pattern with the agent LGWF-net, R_i, discovered from L_{Λ_i}.

Algorithm 1: Compositional discovery

Input: L — an event log over $\Lambda = \Lambda_1 \cup ... \cup \Lambda_k \cup In$,
$IP = A_1 \circledast A_2 \circledast ... \circledast A_k$ – an interface pattern
Output: S — a multi-agent system LGWF-net

$S \leftarrow \mathbb{IP}$
foreach $\Lambda_i \subseteq \Lambda$ **do**
  $R_i \leftarrow$ DISCOVER(L_{Λ_i})
end
$\mathfrak{R} \leftarrow \{R_1, R_2, ..., R_k\}$;
foreach $R_i \in \mathfrak{R}$ **do**
  if *isREFINEMENT*(R_i, A_i) **then**
    REPLACE(S, A_i, R_i)
  end
end

If all agent LGWF-nets, discovered from sub-logs, are proper refinements of the corresponding parts in the interface pattern IP, we will obtain a complete multi-agent system model S where every A_i is successfully replaced with R_i.

However, it is also possible that only some LGWF-nets, discovered from sub-logs, are proper refinements of an interface pattern. For instance, given $IP = A_1 \circledast A_2$, we may obtain that R_1 is not a refinement of A_1, while R_2 is a refinement of

A_2. Then a pattern will only be partially refined, and a system model $S = A_1 \circledast R_2$ will be an *approximation* of the model sought for. In addition, if none of the GWF-nets, discovered from sub-logs, are proper refinements of the interface pattern, then this algorithm will not change an interface pattern. In these cases, we may recommend to modify IP or to develop a new interface pattern.

The correctness of Algorithm 1 is justified by the fact that a multi-agent system LGWF-net S is sound and perfectly fits an event log L, provided that all agent LGWF-nets can be mapped on an interface pattern. Further, we present a collection of interface patterns, which preserve the soundness of agent LGWF-nets R_i, ad formalize and prove the correctness properties of Algorithm 1.

4.2 Interface Patterns

Proper specification of interfaces plays a significant role in establishing the correctness of the compositional process discovery. As mentioned in Introduction and in Section 2, it is easy to arrange agent interactions that will result in deadlocks. Consider, for example, the AS-composition $N_1 \circledast N_2$ shown in Fig. 37. Two agent LGWF-nets N_1 and N_2 are sound in isolation. However, their AS-composition is no longer sound since N_2 may decide not to receive a message from channel c. As a result, N_1 will not be able to synchronize further with N_2.

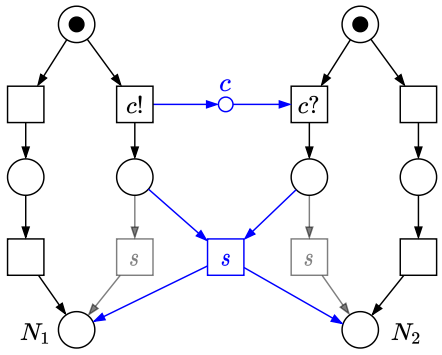


Figure 37: Arbitrary interfaces can result in deadlocks

That is why we do not consider arbitrary interfaces in the compositional process discovery. We design specific *interface patterns* — ready-to-use interface models describing *typical* and basic ways of agent interactions. Moreover, interface patterns preserve the soundness of agent LGWF-nets discovered from filtered sub-logs.

Firstly, we consider the classification and informal (textual) representation of patterns. Secondly, interface patterns are specified formally, via the AS-composition of LGWF-nets.

4.2.1 Classification

Patterns have been traditionally used in software engineering, e. g., *software design patterns* [56]. W. van der Aalst et al. [57] first introduced *workflow patterns* in Business Process Management to consolidate recurrent scenarios in the control-flow of business processes. Later, A. Barros et al. [9] generalized workflow patterns to model *typical* service interactions in large-scale information systems. We take their classification of patterns and recall it below.

Firstly, interface patterns are distinguished by the number of interacting parties:

- *bilateral* patterns specifying interactions between two agents;
- *multilateral* patterns specifying interactions among three or more agents.

Also, interface patterns are classified according to the way agents interact:

- *single transmission* patterns;
- *multiple transmission* patterns.

The number of transmissions corresponds to the number of times interacting agents can exchange messages. Multiple transmission patterns imply repeated message exchange that should have a possibility to be terminated to preserve the soundness of agent behavior.

Within the compositional process discovery, an interface pattern should include three main parts indicated by the scheme shown in Fig. 38. They are:

1. The number of interacting agents.
2. How agents exchange messages via channels.
3. How and when agents execute synchronous actions.

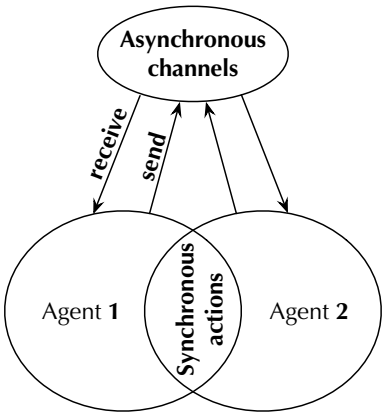


Figure 38: Components of an interface pattern

Following this scheme will allow us to effortlessly translate the informal description of an interface pattern into a composition of LGWF-nets.

While describing interface patterns, we also follow the general principles in the component-based design of information systems [58]:

1. An interface should provide enough information to establish correct communication among agents.
2. An interface should not expose the internal behavior of agents not required for their communications.

4.2.2 Informal representation

We design a set of interface patterns describing asynchronous and mixed asynchronous-synchronous interactions among agents, such that the soundness of their behavior is preserved. Using specific synchronous patterns in isolation is not of

great value in modeling systems with complex agent interactions. That is why we will further consider different combinations of asynchronous message exchange and synchronizations.

Table 4 and Table 5 give an informal representation of twelve interface patterns we use to represent the interaction-oriented architecture viewpoints of a multi-agent system in the compositional process discovery. A pattern contains dummy agent names and crucial aspects of agent interactions, according to the scheme from Fig. 38. We use the short identifiers to refer to these interface patterns further throughout the text.

Table 4: Description of asynchronous interface patterns

Pattern	**ID**	**Description**
Send (Receive)	IP-1	An agent X sends (receives) a message to (from) an agent Y.
Concurrent Send (Receive)	IP-2	An agent X concurrently sends (receives) several messages (>1) to (from) an agent Y.
Alternative Send (Receive)	IP-3	An agent X sends (receives) exactly one out of two (or more) alternative message sets to (from) an agent Y.
Exchange	IP-4	An agent X sends a message to an agent Y. Subsequently, Y sends a response to X.
Concurrent Exchange	IP-5	An agent X concurrently sends several messages (>1) to an agent Y. Then Y sends a response to each message received from X.
Alternative Exchange	IP-6	An agent X sends exactly one out of two (or more) alternative message sets to an agent Y. Subsequently, Y sends a corresponding response to a message received from X.
Multiple Exchange	IP-7	An iterative implementation of IP-4, such that the message exchange continues till an Agent X does not need responses from an Agent Y.
Racing incoming messages	IP-8	An agent X receives one among a set of messages incoming from two or more other agents.

Table 4 considers interface patterns developed using service interaction pat-

terns presented in [9]. Single transmission patterns, IP-1, IP-2, and IP-3, describe rather primitive agent interactions since a sending agent is not supposed to receive an acknowledgment from the other agent. Various ways of asynchronous message exchange are given in patterns IP-4, IP-5, and IP-6. Interface pattern IP-7 describes multiple transmission interactions when one agent can decide to stop the exchange by sending a corresponding message to the other agent.

Multilateral interactions among three or more agents are described in IP-8. According to the specification of this pattern, one of the agents expects to receive one of several messages incoming from the other agents. Sending agents should be properly notified whether their messages are received.

Table 5 describes mixed interface patterns. They combine asynchronous and synchronous agent interactions. Patterns IP-9 and IP-10 extend pattern IP-4 such that agents synchronize either before or after messages are exchanged. Pattern IP-11 extends pattern IP-5 such that agents synchronize and exchange messages concurrently. Pattern IP-12 allows agents to either execute a synchronous activity or exchange messages. This corresponds to an extension of pattern IP-6.

Table 5: Description of mixed interface patterns

Pattern	**ID**	**Description**
Sync Before Exchange	IP-9	Before exchanging messages, agents X and Y execute a synchronous action.
Sync After Exchange	IP-10	Agents X and Y execute a synchronous action after they exchange messages.
Sync And Exchange	IP-11	Concurrently with message exchange, agents X and Y execute a synchronous action.
Sync Or Exchange	IP-12	Agents X and Y either execute a synchronous action or exchange messages but not both.

In the following section, we translate these informal descriptions of interface patterns into the AS-composition labeled GWF-nets.

4.2.3 Formal specification

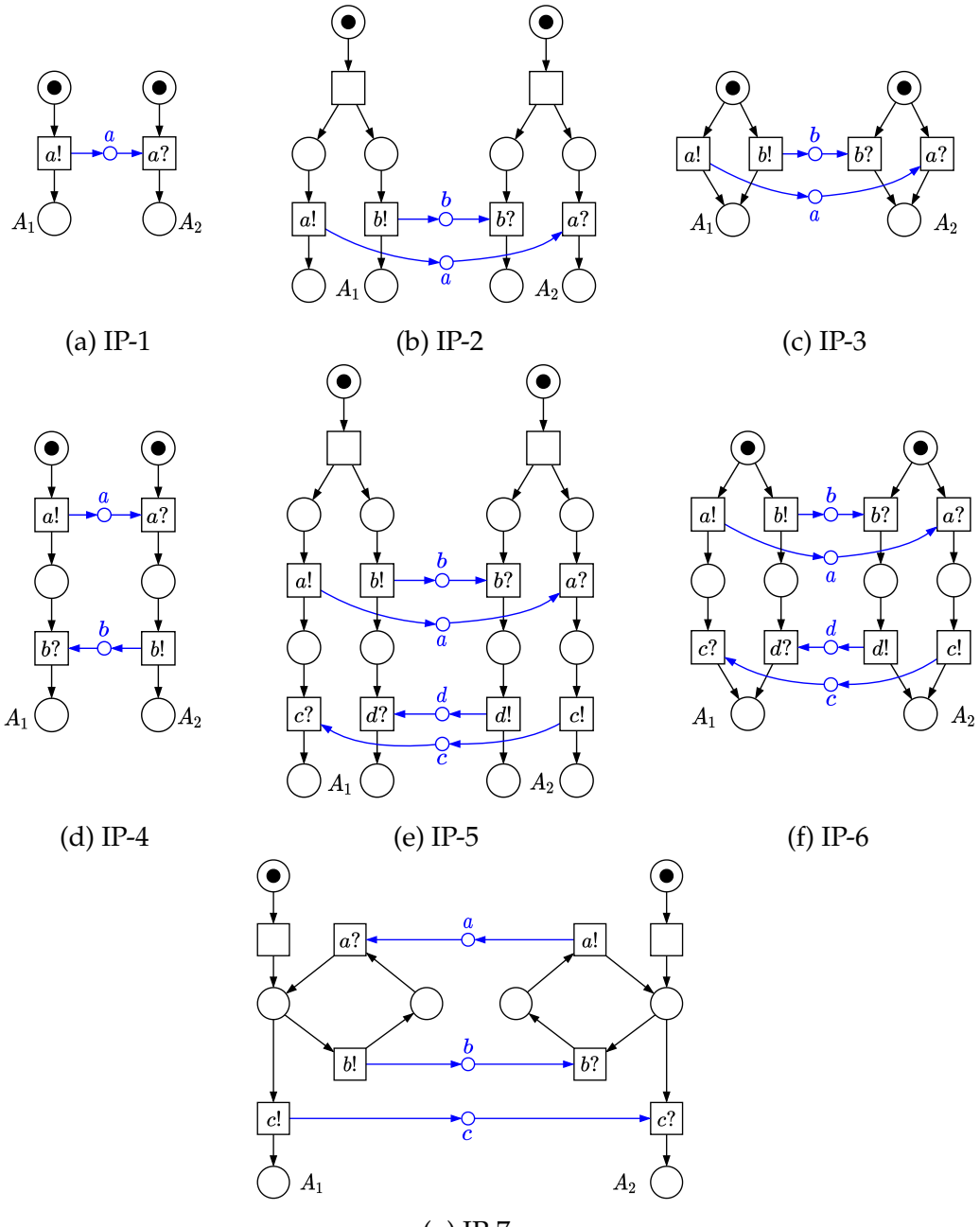
Figure 39 and 40 provide eight LGWF-nets constructed according to the informal description of eight asynchronous interface patterns. Figure 41 provides four LGWF-nets constructed according to the description of four interface patterns, combining both asynchronous and synchronous interactions. We discuss some important features of these models in more detail below.

Every interface pattern is a composition of LGWF-nets representing abstractions of agent behavior. For example, in the LGWF-net of pattern IP-1 shown in Fig. 39a, abstract representations of agent behavior, A_1 and A_2, contain a single labeled transition used to send/receive a message. However, abstractions of agent LGWF-nets can also contain transitions not labeled by interacting actions. They are required to model the specific control-flows of agents. For instance, the LGWF-nets of patterns IP-2, IP-5, IP-8, and IP-11 (see Fig. 39b, Fig. 39e, Fig. 40, and Fig. 41c correspondingly) contain transitions used to model the splits and joins of parallel branches in agent behavior.

In the LGWF-nets of single transmission interface patterns, IP-1, IP-2, and IP-3, channels are added only "in a single direction" to send/receive messages. According to the specification of these patterns, acknowledgments are not expected.

The remainder of bilateral asynchronous interface patterns contains different message exchange variations involving one channel to send a message and the other channel to send an acknowledgment. For example, in the LGWF-net of pattern IP-5 shown in Fig. 39e, there are two concurrent message exchanges between A_1 and A_2. A_1 sends a message to A_2 via channel a, and A_2 sends an acknowledgment to A_1 via channel c. Channels b and d are used similarly.

Consider the LGWF-net of the multilateral pattern IP-8 with three agents, A_1, A_2, and A_3, as shown in Fig. 40. It has the most sophisticated structure among all asynchronous interface patterns. However, it has a clear interpretation. According to the specification of pattern IP-8 from Table 4, A_2 expects to receive one of two messages incoming from A_1 and A_3 through channels a and b correspondingly.



(a) IP-1 (b) IP-2 (c) IP-3

(d) IP-4 (e) IP-5 (f) IP-6

(g) IP-7

Figure 39: Bilateral asynchronous interface patterns: LGWF-nets

Depending on which message is received (the one sent by A_1 or by A_3), A_2 executes the following actions:

1. It notifies A_1 (A_2) by sending an acknowledgment through channel ackA (ackB) that the corresponding message is received;
2. It notifies the agent whose message is not received by sending a message to channel aR or channel bR.

A message left in channel a (b) is removed by the sending agent A_1 (A_3) to preserve the soundness of agents.

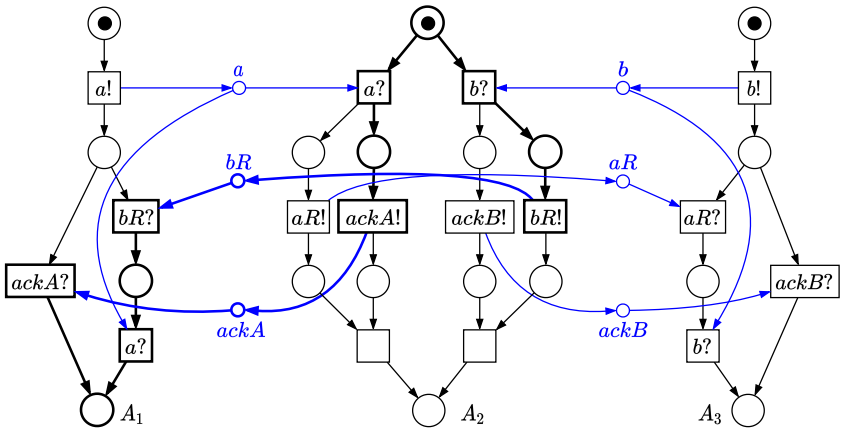
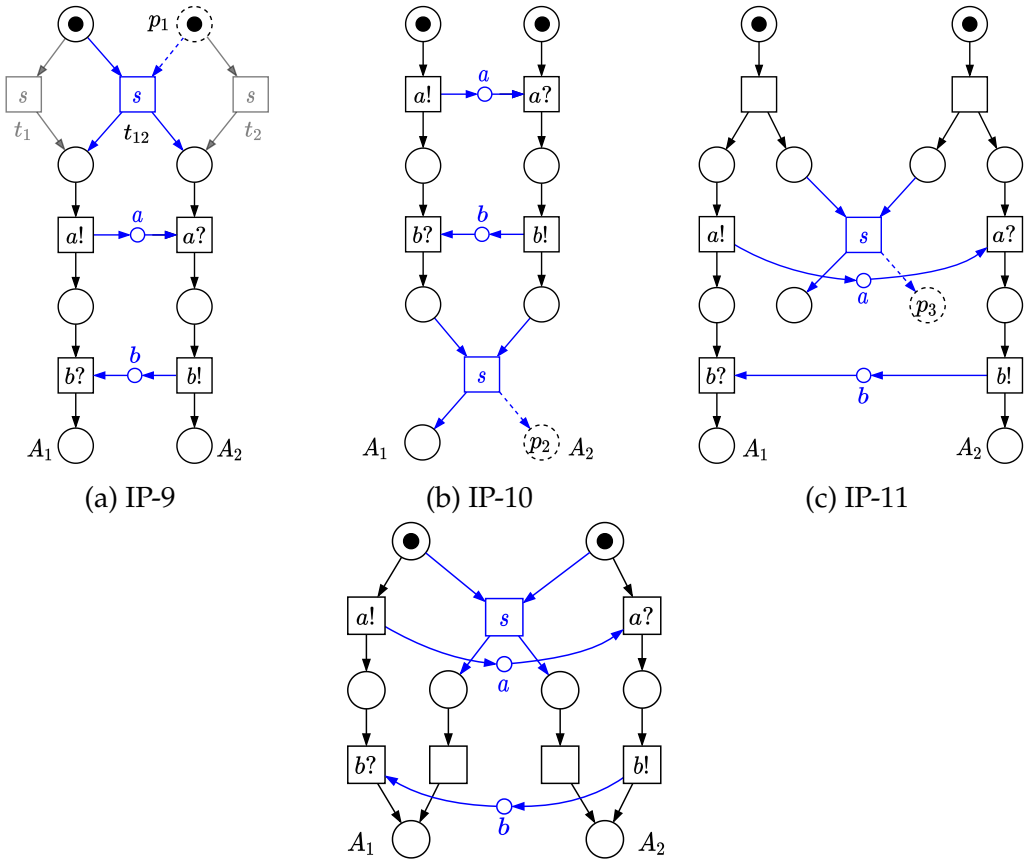


Figure 40: Multilateral asynchronous interface pattern IP-8

In addition, a bold subnet in Fig. 40 corresponds to one of several sequential components covering the GWF-net of pattern IP-8. A similar analysis on sequential components can be done for all LGWF-nets provided in Fig. 39 and Fig. 41.

As discussed earlier, we extend asynchronous interface patterns by introducing synchronizations into the structure of corresponding GWF-nets. In the LGWF-nets of patterns IP-9 and IP-10 shown in Fig. 41a and Fig. 41b, synchronous action s is added before and after the message exchange via channels a and b to extend pattern IP-4. Also, in the LGWF-nets of patterns IP-11 and IP-12 shown in Fig. 41c and Fig. 41d, synchronous action s replaces one of two branches of the message exchange, initially present in asynchronous interface patterns IP-5 and IP-6.

It is also important to note that pair-wise fusion of transitions can lead to *redundant* places that cannot be distinguished by their neighborhoods. Redundant places are identified in LGWF-nets of patterns IP-9, IP-10, and IP-11. They are highlighted by dashed circles: place p_1 in Fig. 41a, place p_2 in Fig. 41b and place p_3 in Fig. 41c. These places can be safely removed to make a corresponding LGWF-net P-simple.



(a) IP-9 (b) IP-10 (c) IP-11

(d) IP-12

Figure 41: Mixed asynchronous-synchronous interface patterns: LGWF-nets

The interface patterns discussed in this section describe interactions among agents in a multi-agent system such that the soundness of agent behavior is not violated. Therefore, Proposition 13 holds.

Proposition 13: Interface pattern LGWF-nets preserve soundness of agents

Interface pattern LGWF-nets IP-1, IP-2, ..., IP-12 are sound.

The proof of Proposition 13 is the straightforward verification of the requirements imposed by Definition 8. Note that the collection of interface patterns presented above is incomplete. One may further extend it, provided that an extended version of Proposition 13 holds for the new patterns as well.

4.3 The First Correctness Theorem

Here, we show that GWF-nets discovered using Algorithm 1 can replay all traces in these event logs. In other words, a GWF-net discovered from an event log L of a multi-agent system by Algorithm 1 *perfectly fits* L.

For what follows, L denotes an event log of a multi-agent system over $\Lambda = \Lambda_1 \cup \Lambda_2 \cup ... \cup \Lambda_k \cup \mathbb{In}$. We need to formalize a "*perfectly fits*" relation between a GWF-net and an event log.

Definition 17: Perfectly fits

Let $N = (P, T, F, m_0, m_f, h, k)$ be an LGWF-net. N *perfectly fits* event log L if and only if $\forall \sigma \in L \, \exists w \in FS(N): \ \sigma = h(w)$.

Then we prove that an LGWF-net, discovered from an event log L of a multi-agent system using Algorithm 1, inherits the perfect fitness of an interface pattern and agent LGWF-nets, discovered from agent sub-logs L_{Λ_i} with $i = 1, 2, ..., k$.

Theorem 3: Perfect fitness preservation

Let $IP = A_1 \circledast A_2 \circledast ... \circledast A_k$ be an interface pattern with a transition labeling function $h_{IP}: T_{IP} \to In \cup \{\tau\}$. Let R_i be a refinement of A_i with a transition labeling function $h_i: T_i \to \Lambda_i \cup \{\tau\}$ for all $i = 1, 2, ..., k$. Let $L \in \mathcal{B}(\Lambda)$ be an event log. If IP perfectly fits L_{In} and R_i perfectly fits L_{Λ_i} for all $i = 1, 2, ..., k$,

then $S = R_1 \circledast R_2 \circledast ... \circledast R_k$ with a transition labeling function $h: T \to \Lambda \cup \{\tau\}$ perfectly fits L.

Proof. The proof is done by contradiction. Assume S does not perfectly fit L. Then $\exists \sigma \in L$, s.t. $\nexists w \in FS(S): h(w) = \sigma$. Since IP perfectly fits L_{In}, $\exists w_{IP} \in FS(IP): h_{IP}(w_{IP}) = \sigma|_{In}$, because L_{In} is a log projection of L on $In \subseteq \Lambda$. Since R_i perfectly fits L_{Λ_i}, $\exists w_i \in FS(R_i) = h_i(w_i) = \sigma|_{\Lambda_i}$, because L_{Λ_i} is a log projection of L on $\Lambda_i \subseteq \Lambda$ for all $i = 1, 2, ..., k$. It is evident that both w_{IP} and w_i (for all $i = 1, 2, ..., k$) are projections of a firing sequence $w' \in FS(S)$ on transitions labeled by In and Λ_i, respectively. Since $\Lambda = \Lambda_1 \cup ... \cup \Lambda_k \cup In$ and taking the above into account, we have that $h(w') = \sigma$. It contradicts the assumption that $\nexists w \in FS(S): h(w) = \sigma$. Hence LGWF-net S perfectly fits L. □

An immediate corollary of Theorem 3 gives the first correctness characteristic of Algorithm 1 as follows.

Corollary 5: The first correctness property of Algorithm 1

LGWF-net S discovered from the event log L of a multi-agent system using Algorithm 1 perfectly fits L.

4.4 The Second Correctness Theorem

Using the formal framework behind the AS-composition of LGWF-nets, we prove that LGWF-nets of multi-agent systems discovered by Algorithm 1 are sound.

By Theorem 2, the soundness of an LGWF-net composition $N_1 \circledast N_2$ is preserved when one of the two LGWF-nets N_1 (N_2) is replaced by its sound refinement R_1 (R_2). R_1 and R_2 are obtained by applying a sequence of refinement rules to N_1 and N_2, respectively.

Recall that, by Corollary 4, a refinement of a sound GWF-net, constructed by applying refinement rules (see Definition 16), is also a sound GWF-net. Then we prove that an LGWF-net S discovered from an event log L of a multi-agent system using Algorithm 1 preserves the soundness of an interface pattern and of agent LGWF-nets discovered from log projections.

Theorem 4: Soundness preservation

Let $IP = A_1 \circledast A_2 \circledast ... \circledast A_k$ be an interface pattern, s. t. A_i is a sound LGWF-net with $i = 1, 2, ..., k$. Let R_i be a refinement of A_i with $i = 1, 2, ..., k$. If IP is sound, then $S = R_1 \circledast R_2 \circledast ... \circledast R_k$ is also sound.

Proof. By Definition 16, since R_i is a refinement of A_i, there is a subsequent application of transformations $\rho_1, \rho_2, ..., \rho_n$ leading from A_i to R_i, i.e., $A_i \xrightarrow{\rho_1} ... \xrightarrow{\rho_n} R_i$, s.t. R_i is sound (by Corollary 4). Then there is an α-morphism $\varphi_i \colon R_i \to A_i$. AS-composition of LGWF-nets is also an LGWF-net. Let $IP' = A_1 \circledast ... \circledast A_{k-1}$, then $IP = IP' \circledast A_k$. By Theorem 2, since IP is sound and there is an α-morphism $\varphi_k \colon R_k \to A_k$, $IP' \circledast R_k$ is also sound. AS-composition of LGWF-nets is commutative. Let $IP'' = A_1 \circledast ... \circledast A_{k-2} \circledast R_k$. Then $IP' \circledast R_k = IP'' \circledast A_{k-1}$. Since $IP'' \circledast A_{k-1}$ is sound and there is an α-morphism $\varphi_{k-1} \colon R_{k-1} \to A_{k-1}$, $IP'' \circledast R_{k-1}$ is also sound. Applying this consecutive construction further, we will arrive to the conclusion that $S = R_1 \circledast R_2 \circledast ... \circledast R_k$ is sound. □

Alternatively, it is possible to note that Theorem 4 is a generalization of Corollary 1 An immediate corollary of Theorem 4 provides the second correctness characteristic of Algorithm 1 as follows.

Corollary 6: The second correctness property of Algorithm 1

LGWF-net S discovered from the event log L of a multi-agent system using Algorithm 1 is sound.

4.5 Related Works: Process Discovery

As mentioned in the Introduction, there are many algorithms for the automated discovery of process models from event logs. Among the most widespread ones are the following: Inductive miner [6], Fuzzy miner [59], Heuristic miner [60], ILP-based (integer linear programming) miner [61], Region Theory-based miner [62], and Genetic miner [63]. A. Augusto et al. [2] conducted a comprehensive and systematic review of these and other process discovery algorithms. The existing process discovery algorithms can tackle different problems connected with the representation of event data in logs. They include *noise*, e. g., the wrong ordering of logged actions, duplicate or missed actions, and *incompleteness*, i. e., an event log represents only a finite fragment of all possible execution sequences.

In general, the synthesis of Petri nets from low-level behavioral representations, e.g., transition systems, is a well-known problem that is to decide whether a given transition system is isomorphic to the reachability graph of some Petri net and then to construct this net [64]. Region theory [65] is the main formal tool used to solve the Petri net synthesis problem. It undoubtedly found use in process discovery when an event log is represented by a finite transition system.

Conformance checking [5] is an essential part of process mining along with process discovery. It is aimed to assess the quality of process models discovered from event logs since different algorithms yield different process models, and they are to be compared. The four main quality dimensions in process discovery are fitness, precision, generalization, and simplicity. J. Buijs et al. [66] discussed the role of these dimensions. The review [2] also indicated the lack of universal measures of the fitness and precision applicable to a wide range of process discovery algorithms. W. van der Aalst [67] discussed the same question from a slightly different perspective, focusing on the urgent need for the consistent requirements of quality measures since there is a significant increase in the use of process discovery algorithms in commercial software tools.

The problem of discovering structured process models from event logs is not

entirely new. Researchers studied this problem in several contexts. In general, there exists the continuum of process models: from *Spaghetti* (poorly structured models) to *Lasagna* (models with a clear structure). Subtle differences between well-structured, structured, and semi-structured process models are hard to formalize [1]. For example, process models discovered by the Inductive miner are called *well-structured* since they are recursively constructed from building blocks. They correspond to the basic constructs such as sequential, parallel, alternative, and cyclic executions.

Researchers offer different techniques to improve the structure of discovered models, e.g., in [68], and to produce already well-structured process models [69–71]. Compositional approaches to improving the structure of discovered process models were proposed as well. A. Kalenkova et al. [72] showed how to discover a readable model from an event log by decomposing the extracted transition system. A. Kalenkova and I. Lomazova [73] studied an advanced technique to deal with cancellations — "exceptional" behaviors — in the process execution and to produce clear and structured process models. In addition, W. van der Aalst et al. [74] proposed an approach for the compositional process discovery based on localizing events using region theory to improve the overall quality of discovered process models. A method for compositional modeling and discovery of structured object-centric Petri nets was proposed by W. van der Aalst and A. Berti in [75], where they used special transition fusions. M. Stierle et al. [76] discussed some design principles of discovering comprehensible models from event logs. They defined metrics estimating the extent to which a discovered model meets these principles. In our study, the clear architecture-aware structure of multi-agent system models results from the independent discovery of process models for interacting agents from log projections.

Within the compositional approach to discovering process models of multi-agent systems, we assume that experts provide specifications of agent interactions in advance. Identifying an interface model from a raw event log of a multi-agent

system is another task that is out of the scope of this paper. We designed a collection of specific interface patterns using typical service interaction patterns studied by A. Barros et al. in [9]. They provide generic solutions to the specification of complex component interactions in large-scale systems. G. Decker et al. [77] and D. Campagna et al. [78] discussed the practical application of interaction patterns to construct corresponding BPMN process models. The correctness of interface patterns was also studied by G. Decker et al. in [79] and by W. van der Aalst et al. in [80]. They formalized patterns using process algebras and open Petri nets — a class of Petri nets with distinguished input and output places. The authors used operating guidelines to construct services correctly interacting with the given one. In our case, an interface pattern comprises highly abstract representations of all interacting agents. Moreover, since interface patterns are known to be correct, they can be reused for all properly constructed refinements, representing the concrete behavior of agents.

4.6 Conclusions of Chapter 4

In this chapter, we described the main algorithm for the compositional discovery of architecture-aware and sound process models from event logs of multi-agent systems. The structure of an architecture-aware model is self-explanatory, i. e., it explicitly shows the behavior of individual agents and their interactions. Firstly, we filter event logs by actions belonging to each agent and obtain a set of sub-logs. Secondly, agent models are discovered from these sub-logs with the help of an existing process discovery algorithm. Finally, we check whether there is a mapping of agent models to the corresponding parts in an interface.

Arbitrary interfaces are not considered since it is easy to arrange agent interactions leading to a deadlock. We designed a collection of specific *interface patterns* describing typical agent interactions. The set of presented interface patterns is based on service interaction patterns studied earlier. It can also be extended with

new models of agent interactions, provided that each new pattern is sound. If a map from agent models towards the corresponding parts in an interface pattern exists, then we can replace this part in a pattern with a discovered agent model. As a result, we obtain an architecture-aware model of a multi-agent system if all agent models are successfully mapped on an interface pattern. Otherwise, if only some agent models can be mapped on an interface pattern, we construct an approximation of a multi-agent system model. In this case, an interface pattern needs to be modified, such that all agent models can be mapped on it.

The correctness of the proposed algorithm for discovering process models from event logs of multi-agent systems is formally demonstrated from two perspectives. Firstly, a model of a multi-agent system inherits perfect fitness of an interface pattern and of agent models discovered from filtered sub-logs. Secondly, a model of a multi-agent system preserves soundness of an interface pattern and of agent models discovered from filtered sub-logs. Proof of the algorithm correctness is based on the theoretical backgrounds considered in Chapters 1–3 and on the construction of interface patterns.

In the following chapter, we conduct the experimental evaluation of the compositional process discovery algorithm with respect to the main hypothesis of our study whether the quality of process models discovered from event logs of multi-agent system using Algorithm 1 is improved.

Chapter 5

Experimental Evaluation

This chapter is devoted to the experimental evaluation of the compositional process discovery algorithm. Using interface patterns and refinement transformations, we generate a collection of artificial event logs of multi-agent systems, which are then processed by the compositional process discovery algorithm. According to the main hypothesis of the proposed compositional approach to process discovery, we compare the process models discovered via the standard direct process discovery with those discovered via Algorithm 1. Then the main outcomes from these comparisons are reported and analyzed.

5.1 Layout of Experiments

Experiments have been designed according to the general scheme of the compositional process discovery (see Fig. 3) with two more steps added: *refinement* and *simulation*, as shown in Fig. 42. We have introduced an *artificial* source of event logs, represented by a reference model N_R.

Reference LGWF-nets are refinements of interface patterns. In the following section, we present two algorithms used for the generation of reference models using refinement rules (see Definition 16).

Simulation of a reference model yields an event log of a multi-agent system. This event log will correspond to the assumptions of the compositional process discovery, i.e., (a) every action in a log is assigned an agent and (b) a set of interacting actions is distinguished. An algorithm for simulating models of multi-agent systems is presented in Section 5.3. It also supports simulation of separate agent models with respect to a *declarative* specification of an interface.

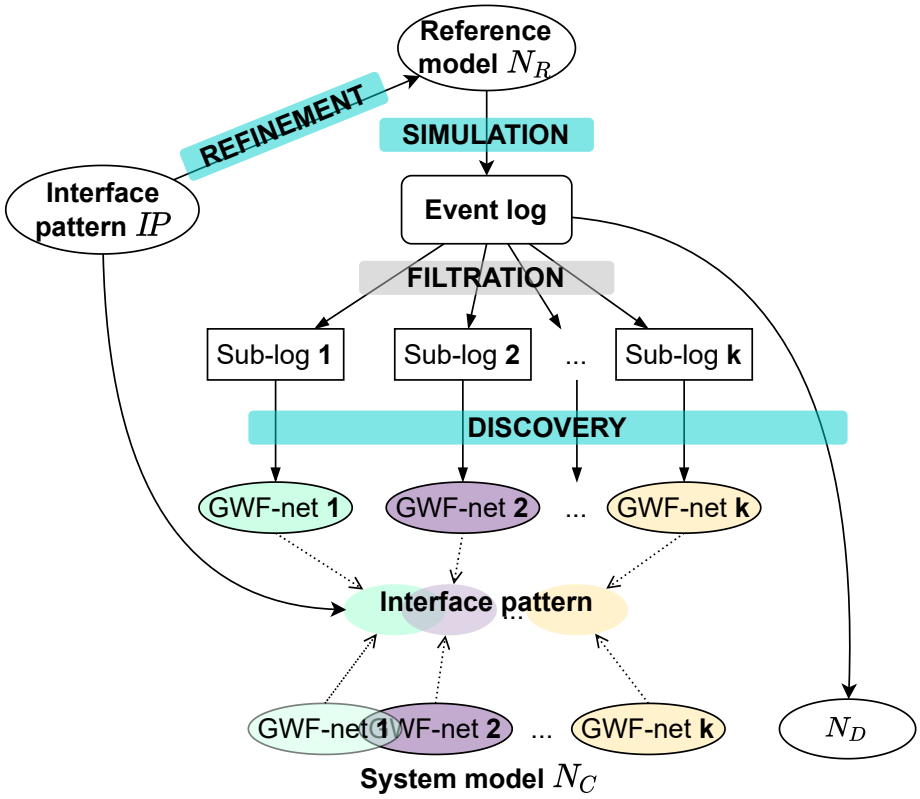


Figure 42: Organization of experiments

This approach based on reference LGWF-nets conforms with a standard way of evaluating process discovery algorithms with the help of so-called *artificial* event logs, which are supposed to exhibit specific characteristics.

Thus, for every interface pattern $\mathrm{IP} = \mathrm{A}_1 \circledast \mathrm{A}_2 \circledast \mathrm{A}_\mathrm{k}$ described in Section 4.2, the procedure specified further has been executed.

Step 1. Construct a sound reference LGWF-net $N_R = N_1 \circledast N_2 \circledast ... \circledast N_k$ where N_i is a refinement of A_i with $i = 1, 2, ..., k$.

Step 2. Simulate the behavior of N_R to obtain an event log L of a multi-agent system with k interacting agents over $\Lambda = \Lambda_1 \cup \Lambda_2 \cup \Lambda_k \cup \mathbb{m}$.

Step 3. Discover a sound LGWF-net N_D *directly* from L.

Step 4. Construct k agent log projections L_{Λ_1}, L_{Λ_2}, ..., L_{Λ_k} (by Definition 2).

Step 5. Discover k sound LGWF-nets N'_1, N'_2, ..., N'_k from agent log projections L_{Λ_1}, L_{Λ_2}, ..., L_{Λ_k}, respectively.

Step 6. Verify whether N'_i. discovered from L_{Λ_i}, is a refinement of A_i in IP, where $i = 1, 2, ..., k$. If so, replace A_i with N'_i and construct $N_C = N'_1 \circledast N'_2 \circledast ...N'_k$.

Step 7. Compare reference LGWF-net N_R prepared at Step 1, directly discovered LGWF-net N_D obtained at Step 3, and compositionally discovered LGWF-net N_C obtained at Step 6.

Construction of reference LGWF-nets and refinement verification (Step 6) are based on Definition 16. Note that if N'_i discovered from the corresponding agent log projection is not a refinement of A_i in the interface pattern IP, then one should consider how to modify the interface pattern preserving its soundness. Thus, Proposition 13 can be extended with new patterns.

Step 3 and Step 5 are explicitly connected with the discovery of process models from event logs. The main requirement, which is imposed on a process discovery algorithm applied here, is that it should produce sound workflow nets. Among the existing algorithms, *Inductive miner*, also mentioned earlier in the text, always discovers sound process models.

The reference, directly and compositionally discovered LGWF-nets are compared (Step 7) using standard and specifically developed *quality* dimensions that are described in Section 5.4.

Finally, the experimental results and corresponding conclusions are presented and discussed in Section 5.5.

5.2 Generation of Reference LGWF-nets

In this section, we describe an approach to the generation of a reference LGWF-net from an interface pattern. A reference LGWF-net is a refinement of an interface pattern, i. e., a result of applying a sequence of refinement rules, described in Section 3.4, to an interface pattern. We consider two ways of constructing a sequence of transformations, *fixed* and *randomized*, that correspond to two algorithms for generating a reference LGWF-net from an interface pattern. These algorithms are described in detail here.

5.2.1 Fixed Generation

Within the fixed generation of reference models, there is a corresponding sequence, $\pi = \langle \rho_1 \rho_2 ... \rho_n \rangle \in RR^*$, of refinement transformations known in advance to be applied to an interface pattern. In other words, the fixed generation is a direct implementation of Definition 16. Algorithm 2 represents this implementation.

Algorithm 2: Fixed generation

Input: LGWF-net $N = (P, T, F, m_0, m_f, h, k)$
Transformations $RT = \{\rho_1, \rho_2, \rho_3, \rho_4\}$
Sequence $\pi = \langle \rho_1, \rho_2, ..., \rho_n \rangle \in RT^*$
Output: LGWF-net $R = (P', T', F', m_0', m_f', h', k')$ – a refinement of N

$R \leftarrow N$
$i \leftarrow 1$
foreach $\rho_i \in \pi$ **do**
  if $\exists X_L' \in P' \cup T'$ *and* ρ_i *is applicable to subnet* $R(X_L')$ *in* R **then**
    $R \leftarrow \rho_i(R, X_L')$
  end
  $i \leftarrow i + 1$
end

If a current rule ρ_i in π can be applied to some subnet generated by a subset of nodes, then we will replace R with the corresponding result of applying ρ to R,

which is initially N. If a current rule ρ_i in π can be applied to different subnets, then the choice is made non-deterministically. Otherwise, if a current rule ρ_i cannot be applied to a subnet in R, we will skip it and pass on to checking the applicability of the next transformation rule in a sequence π.

The correctness of the fixed generation algorithm is easily verified. Firstly, this algorithm always terminates, since the sequence π is finite. Secondly, by Corollary 4, an obtained refinement R preserves the soundness of the initial labeled GWF-net N.

Consider an example, shown in Fig. 43, of applying a sequence of refinement transformations $\pi = \langle \rho_{R4}\rho_{R1}\rho_{R4}\rho_{R2}\rho_{R3} \rangle$ to the interface pattern IP-9 (see Fig. 41a). The choice of subnets transformed by the refinement rules has been made non-deterministically.

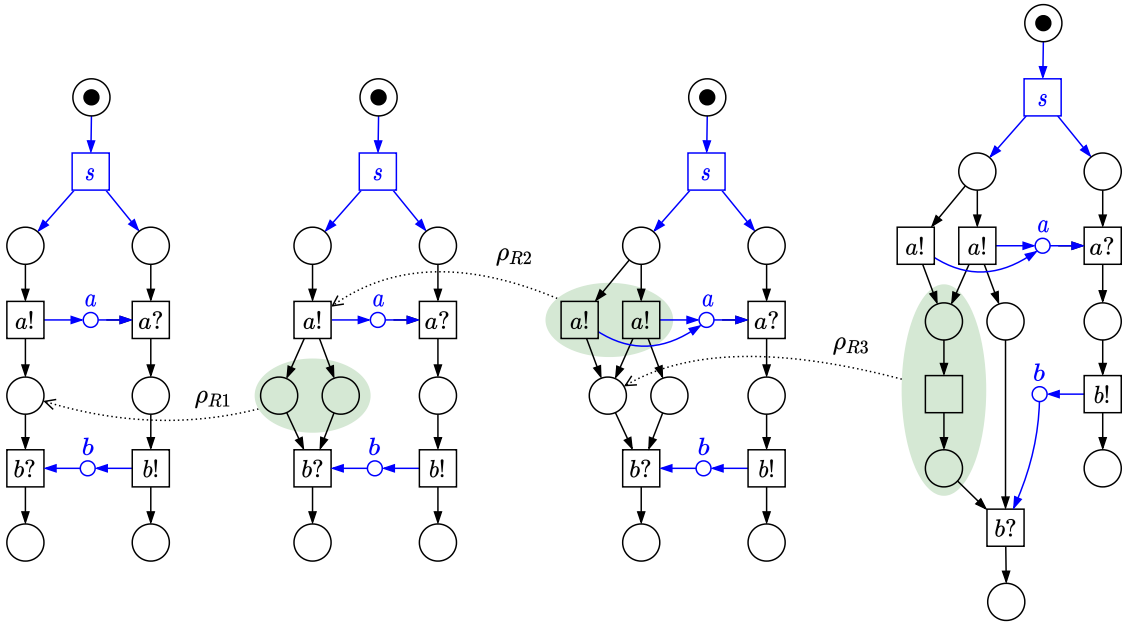


Figure 43: Fixed refinement of an interface pattern

Note that ρ_{R4}-elements in π was not applied since the applicability constraints were not satisfied. Thus the actual sequence of refinement rules that was applied to the LGWF-net of interface pattern IP-9 is reduced to $\langle \rho_{R1}\rho_{R2}\rho_{R3} \rangle$.

5.2.2 Randomized Generation

Within the randomized generation of reference LGWF-nets (see Algorithm 3), a sequence of refinement rules is not known in advance, as opposed to the fixed generation. A specific sequence of refinement rules to be applied to an interface pattern is built according to the parameters defined by a user.

The parameters of the randomized generation include:

1. The *maximum size* of a refinement — the number of places and transitions;
2. The *maximum number of steps* — the number of applied transformations;
3. The *probability* of choosing a specific refinement transformation — the value in the interval $[0, 1]$.

Probabilities are set for all refinement rules, s.t. the sum of all individual probabilities is equal to 1. Below we describe how the specific refinement rule is chosen at every step of Algorithm 3.

Firstly, we compute a set of refinement rules ApR that can be applied to a given LGWF-net R (initially, N) according to the applicability constraints discussed in Section 3.4 (function FindApplicable(R, RR). Secondly, we normalize the probabilities of the applicable refinement rules in ApR and obtain the values of a new $prob'$ function. Afterwards, by generating a random number r, a specific refinement transformation ρ_{Ri} is chosen. Intuitively, the interval $[0, 1]$ is divided into $|ApR|$ intervals, according to the cumulative normalized probabilities of applicable refinement rules. Then we check the placement of r, i. e., which interval the value of r belongs to.

The correctness of Algorithm 3 follows form the following: (a) the total number of steps (the actual length of a sequence of refinement rules) is bounded the maximum size of the obtained refinement and by the maximum number of steps that can be done; (b) by Corollary 4, a constructed refinement R preserves the soundness of the initial LGWF-net N.

Algorithm 3: Randomized generation

Input: LGWF-net $N = (P, T, F, m_0, m_h, h, k)$
Transformations $RR = \{\rho_1, \rho_2, \rho_3, \rho_4\}$
Probabilities $prob\colon RR \to [0,1]$, s.t. $\forall \rho \in RT\colon \sum freq(\rho) = 1$
Maximum number of nodes $maxSize$
Maximum number of steps $maxSteps$

Output: LGWF-net $R = (P', T', F', m'_0, m'_f, h', k')$ – a refinement of N

$R \leftarrow N$
$totalSteps \leftarrow 0$
while $|P' \cup T'| < maxSize$ *OR* $totalSteps \leqslant maxSteps$ **do**
  $ApR \leftarrow$ findApplicable(R, RR)
  $sumProb \leftarrow \sum_{\forall \rho \in AT} prob(\rho)$
  foreach $\rho \in ApR$ **do**
    $prob'(\rho) = \dfrac{prob(\rho)}{sumProb}$
  end
  order AT in the descending order of $prob'$;
  $r \leftarrow$ randomNumber$(0, 1)$
  $cumulProb \leftarrow 0$
  $i \leftarrow 1$
  while $cumulProb < r$ **do**
    $cumulProb \leftarrow cumulProb + prob'(\rho_i)$
    $i \leftarrow i + 1$
  end
  $R \leftarrow \rho_i(R, X'_L)$
  $totalSteps \leftarrow totalSteps + 1$
end

We have conducted an extended evaluation of the randomized generation algorithm by configuring probabilities as follows:

1. Equal probabilities of choosing a specific refinement rule ($\rho_{Ri} = 0.25$).
2. Four cases when the probability of choosing one rule (0.67) outweighs the equal probabilities of the other refinement rules (0.11).

The results of the randomized refinement of interface patterns according to

different configurations of probabilities are presented in Table 6. We show the number of places and transitions in the reference model and in the constructed refinements.

Table 6: Randomized refinement of interaction patterns

			$maxSize$=300, $maxSteps = 1000$									
	Reference		$\rho_i = 0,25$		$\rho_1 = 0,67$		$\rho_2 = 0,67$		$\rho_3 = 0,67$		$\rho_4 = 0,67$	
	|P|	|T|	|P|	|T|	|P|	|T|	|P|	|T|	|P|	|T|	|P|	|T|
IP-1	5	2	134	166	234	66	76	224	156	144	141	166
IP-2	12	6	147	153	216	84	66	256	155	145	146	154
IP-3	6	4	154	149	212	88	85	215	154	147	156	144
IP-4	8	4	132	168	217	83	71	229	152	148	144	156
IP-5	18	10	139	163	207	94	78	222	156	145	157	143
IP-6	12	8	107	193	218	83	72	232	158	142	158	143
IP-7	11	8	140	161	190	110	59	256	143	158	85	215
IP-8	24	16	142	158	208	92	81	219	150	151	144	156
IP-9	9	5	145	155	176	124	58	243	149	152	120	186
IP-10	9	5	143	157	205	95	69	231	146	154	163	137
IP-11	13	7	131	173	217	83	82	218	152	148	157	143
IP-12	10	7	143	158	209	92	79	221	147	153	154	161

As can be seen from these results, the number of places and transitions in the obtained refinements is consistent with the probabilities of applying refinement rules. When all refinement rules are equally probable, we don observe notable differences in the number of places and transitions in the corresponding refinements. However, when the place (transition) duplication has the highest probability, we have that the number of places (transitions) significantly outweighs the number of transitions (places) in the refinement. The predominance of local transition introduction (ρ_{R3}) and place split (ρ_{R4}) does not lead to substantial differences in the numbers of places and transitions. The application of ρ_{R4} requires places with two or more input transitions, which may not be present in the reference and in intermediate refinements.

5.2.3 Related Approaches

Here, we briefly consider and compare the other existing approaches to the generation of process models with the fixed and randomized generation of LGWF-nets.

Process Log Generator PLG2 [81], developed by A. Burattin, is a well-known software used for the random generation of process models. It supports different notations, including Petri nets and BPMN. PLG2 generates process models based on randomly generated context-free grammars and parameters such as the maximum model size, the frequencies of standard behavioral patterns, and others. Compared to our approach, PLG2 offers only the fully randomized model generation and guarantees the behavioral correctness of constructed models. However, within our approach, a reference model may have, for instance, deadlocks that will be preserved in its refinement.

The generation of BPMN process models was also considered by Z. Yan et al. in [82]. The authors of this approach allow specifying the parameters such as the size of models, the frequencies of behavioral patterns, the types of activities. Similar to our approach, they also used a collection of initial BPMN models to generate a set of synthetic models.

PTandLogGenerator [83], developed by T. Jouck and B. Depaire, is another tool supporting the randomized generation of process models. It produces so-called process trees, which specify relations among process activities, for example, sequential, alternative, or concurrent. Process trees can be converted to Petri nets. The prime objective of PTandLogGenerator and the previously mentioned PLG2 is to simulate the behavior of randomly generated process models.

An approach to the generation of *benchmarks*, using random step-wise Petri net refinements, was proposed by K. van Hee and Z. Liu in [84]. Within this approach, the authors also defined a set of refinement transformations similar to those used in our study. Based on the proposed transformations, different Petri net classes were identified and studied. It was shown which transformations can be used to generate all Petri nets representing a given class.

5.3 Generation of Event Logs

Reference LGWF-nets constitute the source for the generation of artificial event logs. The behavior of a reference model can be simulated in a standard way using the firing rule of a net system. GENA [85] is a software tool, which supports the generation of an event log by simulating a net system as well ass the assignment of agents to transitions in a model.

In this section, we propose an alternative approach to the generation of event logs by simulating the behavior of a multi-agent system with *asynchronously* interacting agents. A multi-agent system model is represented via a set of k agent LGWF-nets $\{N_1, N_2, ..., N_k\}$. The AS-composition of these nets is not constructed. Asynchronous interactions between agent LGWF-nets are expressed using so-called *interface constraints*. They allow us to specify not only the order of interacting actions, but also between any other pair of observable actions. In other words, the behavior of interacting agents has the *imperative* representation (LGWF-nets), while constraints on their interaction — *declarative* representation (formulae). We design a corresponding generation algorithm, which simulates the behavior of a set of LGWF-nets with respect to interface constraints.

5.3.1 Interface Formulae

Here we consider multi-agent systems with asynchronously interacting agents. Then a set of interacting actions is restricted to operations implemented with asynchronous channels, i.e. sending or receiving a message.

A multi-agent system model consists of k disjoint LGWF-nets $N_1, N_2, ..., N_k$, which represent the behavior of individual agents and the *interface constraints* $\mathcal{I}$ on their asynchronous interactions.

We additionally make the following assumptions:

1. Transitions in agent LGWF-nets have individual labels, i. e., different agents execute different (interacting) agents.

2. Transitions involved into agent interactions does not belong to cycles in agent LGWF-nets. Therefore, they can occur in every execution not more than once.

Asynchronous interface patterns IP-1–IP-6 and IP-8 (see Fig. 39), describing acyclic and asynchronous interactions among agents, meet these assumptions.

Interfaces are defined in terms of positive logical *formulae* over atomic constraints. Formal definitions are discussed further.

Let N_i with $i = 1, 2, ..., k$ be an LGWF-net, s. t. $\mathrm{dom}(h_i) \cap \mathrm{dom}(h_j) = \varnothing$, where $j = 1, 2, ..., k$ and $i \neq j$. Two types of *atomic constraints* are defined, namely $\lambda_1 \lhd \lambda_2$ and $\lambda_1 \overline{\lhd} \lambda_2$, where λ_1 and λ_2 are labels of transitions in two different LGWF-nets, i. e., $\lambda_1 \in \mathrm{dom}(h_i) \Leftrightarrow \lambda_2 \notin \mathrm{dom}(h_i)$.

It is easy to see that a set of k disjoint LGWF-nets is itself an LGWF-net where nodes are the union of all nodes in these k nets. Let $\mathcal{N} = (P, T, F, m_0, m_f, h, k)$ denote an LGWF-net obtained through the union of nodes of k disjoint agent LGWF-nets, and $w \in FS(\mathcal{N})$ be a firing sequence of $\mathcal{N}$. The *validity* of these atomic constraints for a given execution $\sigma = h(w)$ of $\mathcal{N}$ is defined as follows:

1. $\sigma \models \lambda_1 \lhd \lambda_2 \Leftrightarrow$ if λ_2 occurrs in σ, then λ_1 occurss before λ_2 in σ.
2. $\sigma \models \lambda_1 \overline{\lhd} \sigma_2 \Leftrightarrow$ if λ_1 does not occur before λ_2 in σ.

When $\sigma \models \psi$, we say that ψ is *valid* for σ, and σ *satisfies* ψ.

The atomic constraint $\lambda_1 \lhd \lambda_2$ implies that λ_2 should be always preceded by λ_1, e. g., a message can be only if it has already been sent (cf. the construction of the AS-composition in Definition 12). Thus, $\lambda_1 \lhd \lambda_2$ is valid for an execution $\sigma = \sigma_1\lambda_1\sigma_2\lambda_2\sigma_3$ and is not valid for $\sigma = \sigma_1\lambda_2\sigma_2$ if σ_1 does not contain λ_1.

The atomic constraint $\lambda_1 \overline{\lhd} \lambda_2$ implies that λ_2 cannot occur if λ_2 has happened before, e. g., if a message has been already sent by e-mail, it should not be re-sent via fax again. An execution $\sigma = \sigma_1\lambda_2\sigma_2$ satisfies $\lambda_1 \overline{\lhd} \lambda_2$ if σ_1 does not contain λ_1, while $\sigma = \sigma_1\lambda_1\sigma_2\lambda_2\sigma_3$ does not satisfy this constraint.

These two atomic constraints are not negation of one another. Both $\lambda_1 \lhd \lambda_2$ and $\lambda_1 \overline{\lhd} \lambda_2$ are valid for an execution of an LGWF-net that does not contain λ_2.

Thus, a *language* of interface constraints for a multi-agent system $\mathcal{N}$ is defined by the following grammar rules:

$$\begin{aligned} \mathtt{Atom} &::= \lambda_1 \lhd \lambda_2 \,|\, \lambda_1 \overline{\lhd} \lambda_2, \\ \psi &::= \mathtt{Atom} \,|\, \psi_1 \vee \psi_2 \,|\, \psi_1 \wedge \psi_2, \end{aligned}$$

where $\mathtt{Atom}$ is an atomic constraint, and ψ is a constraint formula.

The validity of a constraint formula ψ for an execution σ in $\mathcal{N}$ is defined in a standard way:

$$\begin{aligned} &\sigma \models \psi_1 \vee \psi_2 \Leftrightarrow \sigma \models \psi_1 \text{ or } \sigma \models \psi_2, \\ &\sigma \models \psi_1 \wedge \psi_2 \Leftrightarrow \sigma \models \psi_1 \text{ and } \sigma \models \psi_2. \end{aligned}$$

Let L be an event log of a multi-agent system with asynchronously interacting agents over $\Lambda = \Lambda_1 \cup \Lambda_2 \cup ... \Lambda_k \cup In$, and ψ be an interface constraint formula. Then ψ is valid for L iff ψ is valid for every trace in L.

Interface formulae can express different useful interaction constraints, e. g., formula $\psi = A \overline{\lhd} B \wedge B \overline{\lhd} A$ describes a conflict between actions A and B, i.e., A and B cannot occur together in the same execution.

Recall that a multi-agent system with asynchronously interacting agents consists of $\mathcal{N}$ — an LGWF-net obtained by uniting k disjoint agent LGWF-net, and $\mathcal{I}$ — an interface constraint formula built according to the grammar described above. We denote a multi-agent system by a pair $S = (\mathcal{N}, \mathcal{I})$. An execution of S corresponds to an execution $\sigma = h(w)$ of LGWF-net $\mathcal{N}$, where $w \in FS(\mathcal{N})$, s. t. $\sigma \models \mathcal{I}$. The following proposition is an immediate consequence of the above definitions.

Proposition 14: Execution of a system can be projected on agent behavior

Let $S = (\mathcal{N}, \mathcal{I})$ be a multi-agent system, and σ be an execution of S. Then, for every agent LGWF-net $N_i \in \mathcal{N}$, $\sigma|_{dom(h_i)}$ is an execution of N_i.

Consider the example of a simple multi-agent system with two agents shown

in Fig. 44. Let $\mathcal{I} = (A \overline{\lhd} B) \wedge (B \overline{\lhd} A)$, which means that B should be in conflict with A. Consider an execution $\sigma = x_1 B\, y_2 x_3$ of this system satisfying $\mathcal{I}$. Projecting σ on agent LGWF-nets gives executions $x_1 x_3$ and $B\, y_2$, which are the corresponding executions of these agent nets.

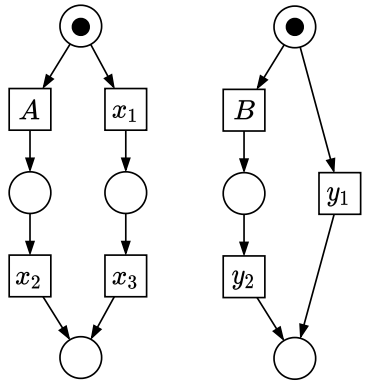


Figure 44: A multi-agent system with two agents

Finally, we present constraint formulae (see Table 7), which correspond to the asynchronous interface patterns IP-1, ..., IP-6, and IP-8, which describe acyclic interactions between agents. Then these formulae can be used to generate event logs by representing a refinement of an interface pattern via a set of agent LGWF-nets where each of them is a refinement of the specific part in a pattern, rather than their AS-composition.

5.3.2 Simulation Algorithm

Interface formulae presented in the previous section define *declarative* restriction on the behavior of a multi-agent system. To simulate its behavior, we define the operational semantics based on a special transition firing rule for selecting and executing the step in the execution of a multi-agent system model. We call this rule an *interface-driven* firing rule to distinguish it from the standard firing rule of net systems. This extended rule should be consistent with the declarative interface constraints on asynchronous interactions among agents.

Table 7: Interface formulae for asynchronous interface patterns

Pattern	Constraint formula
IP-1	$a! \lhd b!$
IP-2	$a! \lhd a? \wedge b! \lhd b?$
IP-3	$a! \lhd a? \vee b! \lhd b?$
IP-4	the same as for IP-2
IP-5	$a! \lhd a? \wedge b! \lhd b? \wedge c! \lhd c? \wedge d! \lhd d?$
IP-6	$(a! \lhd a? \wedge c! \lhd c?) \vee (b! \lhd b? \wedge d! \lhd d?)$
IP-8	$a! \lhd a? \wedge (ackA! \lhd ackA? \vee bR! \lhd bR?) \vee$ $b! \lhd b? \wedge (ackB! \lhd ackB? \vee aR! \lhd aR?)$

Interface-driven firing rule

Let $S = (\mathcal{N}, \mathcal{I})$ be a multi-agent system, where $\mathcal{N} = (P, T, F, m_0, m_f, h, k)$ is an LGWF-net obtained as the union of k disjoint agent LGWF-nets, and $\mathcal{I}$ is an interface formula.

Firstly, we convert $\mathcal{I}$ to a disjunctive normal form (DNF) using standard logical laws. For example, the formula for the interface pattern IP-6 (see Table 7) is already in DNF, while to transform the formula for IP-8, one need to expand the brackets. Then, an interface $\mathcal{I} = \bigvee_{j=1}^{n} C_j$, where $C_j = \bigwedge_{\ell=1}^{m} A_\ell$, and A_ℓ is an atomic constraint. By abuse of notation, we denote by $\mathcal{I}$ also the set of its conjuncts, and by C_j – the set of atomic constraints in a conjunct C_j.

Obviously, an execution σ satisfies $\mathcal{I}$ iff $\exists C_j \in \mathcal{I}\colon \sigma \models C_j$, i. e. σ satisfies at least one conjunct in $\mathcal{I}$. So, to generate a model execution, we need to choose a conjunct C_j and fire transitions in $\mathcal{N}$ only if these firings do not violate C_j.

Then we define $T_{\mathcal{I}} \subseteq T$ to be the set of transitions in $\mathcal{N}$ involved in agent interaction, i.e., $t \in T_{\mathcal{I}}$ iff $h(t)$ occurs in $\mathcal{I}$. We call transitions in the set $T_{\mathcal{I}}$ *interface transitions*.

Independent transitions from $T \setminus T_{\mathcal{I}}$ fire according to the standard firing rule

for Petri nets. Firing interface transitions is restricted by the constraint formula $\mathcal{I}$. To check whether firing of a transition t violates C_j, we keep the current historical execution, i.e., a sequence of already fired transitions. When a transition $t \in T_{\mathcal{I}}$ is enabled according to the standard firing rule at a current marking m, and an atomic constraint $\lambda \lhd h(t)$ occurs in C_j, then t is defined to be enabled only if λ occurs in the current run. Similarly, if $\lambda \overline{\lhd} h(t)$ occurs in C_j, then t is defined to be enabled only if λ does not occur in the current execution.

Thus, the *operational semantics* of a multi-agent system model $S = (\mathcal{N}, \mathcal{I})$, where $\mathcal{N} = (P, T, F, m_0, m_f, h, k)$ and $\mathcal{I} = \bigvee_{j=1}^{n} C_j$ is defined by the following procedure.

Step 1. Choose non-deterministically a conjunct C in $\mathcal{I}$.

Step 2. Start with the initial marking m_0 and the empty sequence ϵ for a current execution σ.

Step 3. For a current marking m and a current execution σ repeat while there are enabled transitions in $\mathcal{N}$:

1. Compute the set T_{ok} of all transitions enabled at m, not violating atomic constraints from C with respect to σ;
2. Choose non-deterministically a transition t from T_{ok};
3. Fire t by changing the current marking to $m' = (m \setminus {}^\bullet t) \cup t^\bullet$, and adding $h(t)$ to the execution σ.

The obtained execution σ corresponds to a trace in an event log L produced by a multi-agent system $S = (\mathcal{N}, \mathcal{I})$. We next discuss an algorithm, which implements the interface-driven firing rule.

To begin with, for each conjunct C occurring in $\mathcal{I}$, we simulate the behavior of S to check if it is possible to obtain an execution σ satisfying C. If we cannot obtain such an execution by simulating S, this conjunct is excluded. As a result, we come to a set of conjuncts $\mathcal{I}' \subseteq \mathcal{I}$ which can actually be satisfied by executions of S or an empty set if $\mathcal{I}$ cannot be satisfied by S. If $\mathcal{I}' = \varnothing$, then the simulation is terminated yielding the empty event log.

That is why we can simulate the behavior of S with respect to conjuncts occurring only in $\mathcal{I}'$. Starting a new iteration of the simulation, we randomly choose a conjunct from $\mathcal{I}'$ and fire transitions in $\mathcal{N}$ according to the interface-driven firing rule.

Algorithm 4 is used for generating a single trace, which satisfies a non-deterministically chosen conjunct C from $\mathcal{I}'$ according to the operational semantics of a multi-agent system discussed above.

Algorithm 4: Single trace generation

```
Input: N = (P, T, F, m_0, m_f, h, k), I', maxSteps
Output: σ, s.t. σ ⊨ I'
σ ← ϵ, m ← m_0, i ← 1
C ← pickRandomConjunct(I')
while (i ⩽ maxSteps) ∧ (m ≠ m_f) do
    T_ok ← findEnabledTransitions(N, m, C, σ)
    if T_ok ≠ ∅ then
        t ← pickRandomTransition(T_ok)
        m ← fireTransition(N, m, t)
        if λ(t) ≠ τ then
            σ ← σ + h(t)
            i ← i + 1
        end
    else
        σ ← ϵ
        break
    end
end
```

Algorithm 5 computes the set of enabled transitions, which do not violate constraints of C. Firstly, we find a set of transitions enabled at a reachable marking m according to the standard firing rule. Secondly, if m enables interface transitions, we check whether the current execution $\sigma = h(w)$, s.t. $m_0[w\rangle m$, satisfies constraints of C using the interface-driven firing rule. An execution σ is a trace to be recorded into an event log L of a multi-agent system S.

Consider an example based on the multi-agent system shown earlier in Fig. 44. Assume $\mathcal{J} = (A \lhd B) \vee (y_1 \lhd x_1 \wedge x_2 \overline{\lhd} y_1)$. $C = y_1 \lhd x_1 \wedge x_2 \overline{\lhd} y_1$ is chosen. We are at the initial marking, so the execution is empty, i.e. $\sigma = \epsilon$. Enabled transitions are $\{A, x_1, B, y_1\}$. However, x_1 cannot fire, since it should wait until y_1 is executed. Then, non-deterministically B fires. Subsequently, the run is $\sigma = B$, and the enabled transitions are $\{A, x_1, y_2\}$, but x_1 still cannot fire. We can choose A to fire. Then, the execution is $\sigma = BA$, and the enabled transitions are $\{x_2, y_2\}$, which are not influenced by C. As a result, we can obtain a trace $\sigma = BAy_2x_2$ satisfying C, and the projections of σ on agent transitions, Ax_2 and By_2, are corresponding executions of agent LGWF-nets.

Algorithm 5: Function findEnabledTransitions

```
Input: N = (P, T, F, m_0, m_f, h, k), m ∈ [m_0⟩, C ∈ J', σ
Output: A set of transitions T_ok enabled w.r.t to C
T_m ← stEnabledTransitions(N, m)
T_ok ← T_m \ T_J
foreach t ∈ T_m ∩ T_J do
    foreach A ∈ C do
        if A = λ ◁ h(t) then
            if σ = uλv then
                T_ok ← T_ok ∪ t;
            end
        else if A = λ ◁̄ h(t) then
            if σ ≠ uλv then T_ok ← T_ok ∪ t;
        end
    end
end
```

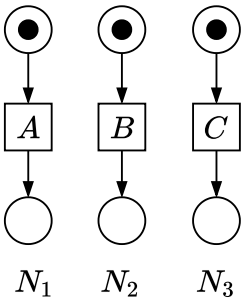
5.3.3 Log Generation Examples

Algorithm 4 and Algorithm 5 have been developed as an extension to GENA [86], which allows users to simulate the behavior of a multi-agent system according to declarative constraints on agent interactions.

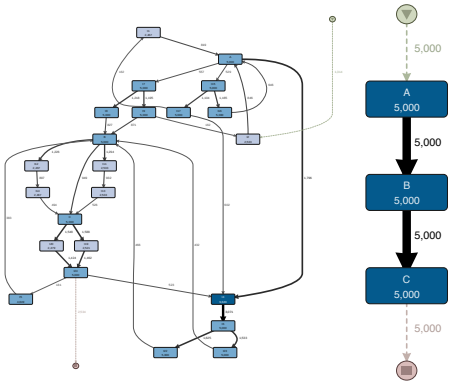
Five use cases have been prepared to demonstrate the capacity of the interface-driven simulation of multi-agent systems' behavior. They are different from the asynchronous interface patterns considered in Section 4.2. For every case, we provide an abstract representation of agent behavior and a visual representation of a generated event log filtered according to interacting actions, s.t. it is clear whether the corresponding interface formula is satisfied.

Sequencing

Consider a system with three interacting agents (see Fig. 45). Each agent always executes one action, which are not refined. We have simulated it with respect to the interface $\mathcal{I} = A \lhd B \wedge B \lhd C$. Intuitively, this interface may mean each agents consecutively prepare a resource needed for the other agent.



(a) system



(b) event log

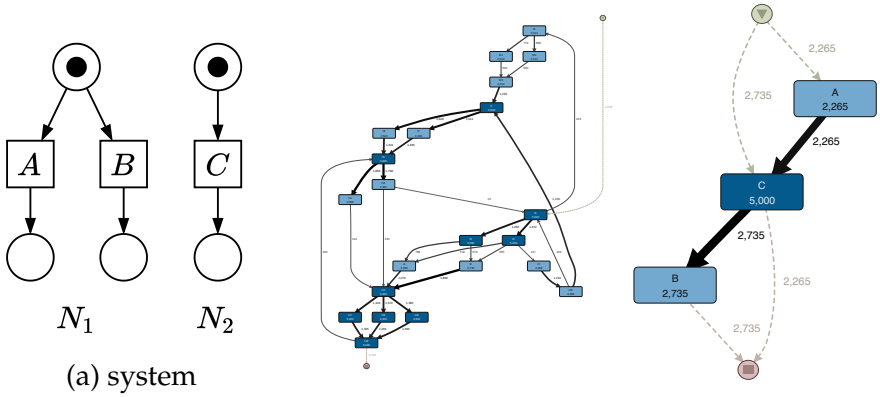
Figure 45: Sequential interaction among three agents

As it can be seen, all 5000 traces in the generated event log satisfies $\mathcal{I}$.

Conditional sequencing

As opposed to sequencing, conditional sequencing allows for several execution options. In this case, a system consists of two agents, one of which has alternative

branches (see Fig. 46). An interface formulae for the conditional sequencing is defined as follows: $\mathcal{I} = A \lhd C \vee C \lhd B$.



(a) system

(b) event log

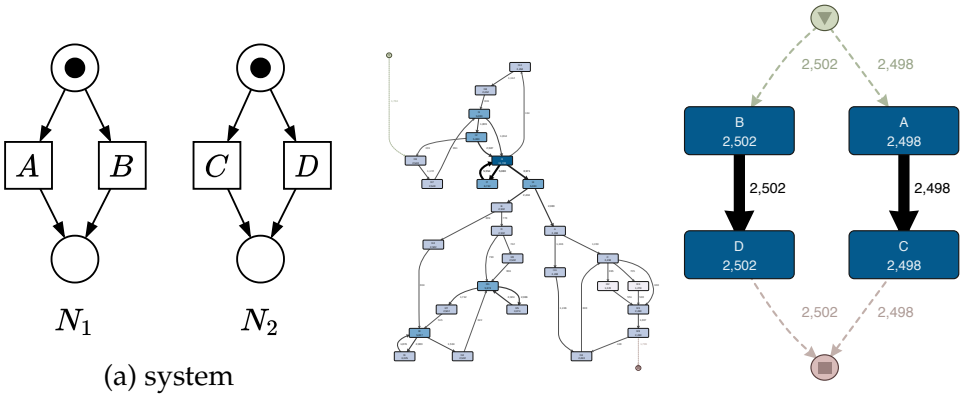
Figure 46: Sequential interaction with options

As it can be seen from Fig. 46, 2735 of all 5000 traces in the generated event log satisfies $C \lhd B$, and the remaining 2265 traces satisfies $A \lhd C$.

Alternative interaction

This case is the direct implementation of the interface pattern IP-3 (see Fig. 39c), modeling the alternative message exchange. A system consists of two interacting agents both having alternative branches (see Fig. 47). An interface formula for this case is as follows: $\mathcal{I} = A \lhd C \vee B \lhd D$.

It is easy to see that the generated event log visualized in Fig. 47 satisfies $\mathcal{I}$. 2502 of all 5000 traces satisfy $B \lhd D$, and the remaining 2498 traces satisfy $A \lhd C$.



(a) system

(b) event log

Figure 47: Alternative interaction a.k.a. interface pattern IP-8 shown in Fig. 39c

Interactions using $\overline{\lhd}$-constraints

Assume a multi-agent system consists of two agents as in the previous case (see Fig. 47). Consider the event log of this system generated according to the interface formula $\mathcal{I} = A \,\overline{\lhd}\, C$ provided in Fig. 48. It is clear from the visualization of the generated event log that C is never preceded by A. Negative $\overline{\lhd}$-constraints allow for a more compact interface specification.

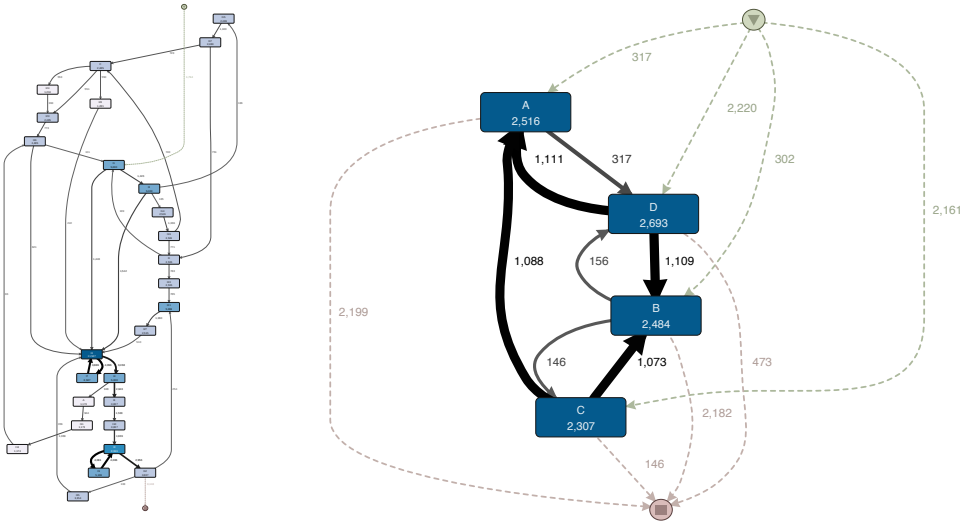
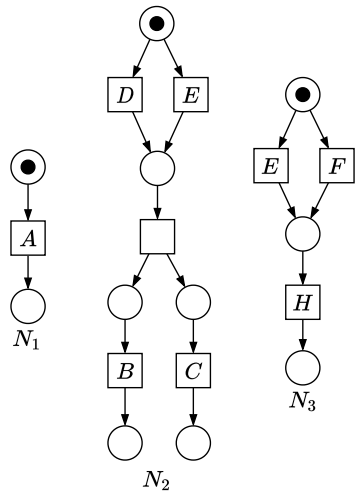


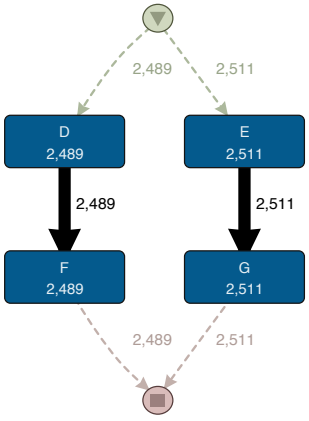
Figure 48: Interactions via negative $\overline{\lhd}$-constraints: event log

Complex interactions

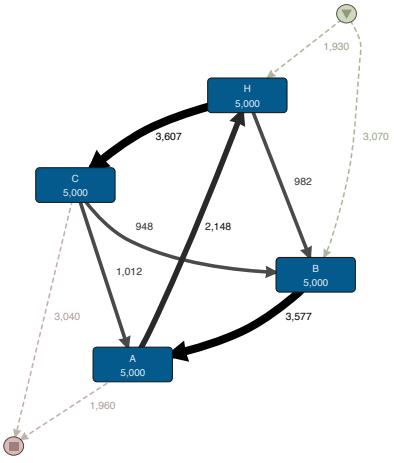
In this final case, we consider mixed interactions among three different agents, as shown in Fig. 49a. The event log, which has been generated according to the interface formula $\mathcal{I} = (B \lhd A) \wedge (H \lhd C) \wedge (D \lhd F \vee E \lhd G)$, is represented in Fig. 49b and in Fig. 49c. The formula is given in a conjunctive normal form for brevity. This event log has been filtered in two ways to verify the satisfaction of $\mathcal{I}$.



(a) system



(b) actions D, E, F, G



(c) actions A, B, C, H

Figure 49: Complex interactions among three agents

The visual representation confirms that the generated event log of a multi-agent system with three agents fully satisfies the initial interface formula.

5.3.4 Related Approaches

Process discovery algorithms employ a variety of heuristics. That is why testing is extensively used to evaluate their efficiency and validity. It is usually performed using both real-life and artificially prepared event logs with appropriate characteristics. The latter are prepared using event log generators.

Process Log Generator PLG2 [81] and *PTandLogGenerator* [83], discussed above within the framework of reference model generation, are also used to record the behavior of process models in event logs.

The main goal of the tools mentioned above is the randomized testing using sets of models and event logs. However, in some cases there is a need to generate event logs from specific process models that have been prepared on the basis of the real data or expert knowledge. If this is the case, one can use the tool *GENA* [85]. It aims at generating sets of event logs from a Petri net model. The approach allows users to use preferences to influence a control-flow and to artificially introduce a randomized noise into an event log. The improved version of *GENA* can generate event logs from BPMN 2.0 models [87]. Most basic BPMN constructs are supported: tasks, gateways, messages, pools, lanes, data objects.

Colored Petri nets can also be used to generate event logs [88]. Authors have developed the extension for *CPN Tools* that can generate randomized event logs based on a given colored Petri net. The main drawback of this approach is that it implies writing *Standard ML* scripts, which leads to possible problems during tool adaptation for a specific task. Moreover, this approach and *GENA* do not support multi-agent systems with independent asynchronous agents.

Declarative process models can also be used to generate event logs [89]. This approach is based on construction of a finite automaton using a *Declare* process model. The tool can generate a specified number of strings accepted by this

automaton. Strings are generated using the automaton and its randomized execution. Afterwards, each string is transformed into a log trace with necessary attributes. This tool is useful, when the only information about the process is the set of constraints. This approach is also not appropriate for the simulation of a multi-agent system, because it does not support the imperative description of individual agents.

Our interface-drive approach to the simulation of multi-agent systems combines the imperative specification of agents via the union of disjoint LGWF-nets and the declarative description of interface constraints. They can be used to specify the order of executing both interacting and local transitions in agent LGWF-nets. The key open question with the interface formulae, planned to be studied in the future, is as follows. Interface constraints allows us to describe only a specific class of asynchronous interactions, while the AS-composition of LGWF-nets does not have these constraints. In this light, we plan to investigate the relationships between the AS-composition and interface formulae, especially in the context of modeling sound interface patterns. We will consider the interface constraints for specifying synchronous and cyclic interactions among agents in a multi-agent system as well as to study the way to express negative $\overline{\lhd}$-constraints via the AS-composition.

5.4 Conformance Checking

Directly and compositionally discovered LGWF-nets (obtained at Step 3 and Step 6 in the experiment plan discussed in Section 5.1, respectively) relate differently to the initial event log, obtained after simulating the behavior of the reference models. The general picture of relations between the traces in an event log L and the executions of an LGWF-net N is given in Fig. 50.

The estimation of the correspondence between an event log and a process model is the main problem in the field of *conformance checking* [5]. In addition, within the conformance checking, the structural complexity of process models is evaluated

as well. There are four main quality dimensions offered in conformance checking: *fitness*, *precision*, *generalization* and *simplicity* [66], briefly discussed in Introduction as well. They are aimed to build a holistic view on the quality of process models discovered from event logs.

In our experiments, we estimate *precision* and *simplicity* of the reference, directly and compositionally discovered LGWF-nets with respect to the artificial event logs, as specified by Step 7 in our experiment plan.

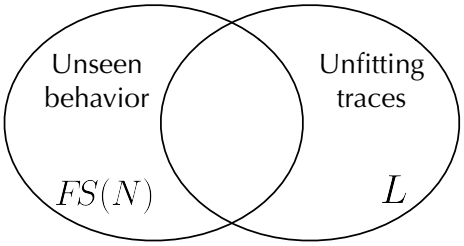


Figure 50: Relation between event log L and LGWF-net N

Fitness is a value in the interval $[0, 1]$ that demonstrates how well a process model can replay every trace from an event log. In the general case shown in Fig. 50, a part of an event log (*unfitting traces*) may not be covered by the firing sequences in a process model. The more the number of unfitting traces in L is, the lower the fitness of a process model is. By Definition 17, a process model *perfectly fits* an event log (fitness $= 1$) if it can execute all traces in this event log, i.e., there are no unfitting traces.

Note that, by Corollary 5, GWF-nets obtained by Algorithm 1 perfectly fit event logs. Apart from that, existing process discovery algorithms allows configuring the desired fitness level. It may be necessary to decrease the fitness, while working with *noisy* and real-life event logs, where there can be missing or duplicate actions, the wrong ordering of actions etc. Artificial event logs do not have such problems. Thus, we do not need to estimate the fitness of reference, directly and compositionally discovered models.

Precision is a value in the interval $[0, 1]$ that evaluates a ratio of the behavior

allowed by a process model and not allowed by an event log (*unseen behavior* as shown in Fig. 50). Perfectly precise models can only replay the traces present in an event log. However, an event log represents only a finite fragment of all possible process executions. That is why perfectly precise models are of very restrictive use. A most wide-spread approach, used in our experiments as well, to the precision estimation, is based on *aligning* the firing sequences of a process model with the traces in an event log [90].

The (structural) complexity of a discovered process model is captured by the *simplicity* dimension. We express the simplicity of a process model through:

1. the number of places, transitions and arcs;
2. the number of *neighboring transitions* between different agents.

Recall that the compositional process discovery aims to synthesize *architecture-aware* process models, the structure of which indicates agent behavior and interactions. Thus, we expect the simplicity to be the main distinguishing feature of compositionally discovered LGWF-nets compared to those discovered directly from event logs of multi-agent systems. Below, we explain the main idea behind the notion of neighboring transitions.

Neighboring transitions

The notion of neighboring transitions is introduced as an attempt to estimate the extent to which a structure of a discovered process model corresponds to the architecture of a multi-agent system with respect to agent interactions. In other words, an architecture-aware model of a multi-agent system explicitly indicates the behavior of individual agents as well as the way they interact by exchanging messages and executing synchronous activities. Precise definitions are discussed below.

Let $L \in \mathcal{B}(\Lambda^+)$ be an event log of a multi-agent system over $\Lambda = \Lambda_1 \cup \Lambda_2 \cup ... \cup \Lambda_k$. Let $N = (P, T, F, m_0, m_f, h, k)$ be an LGWF-net, where $h\colon T \to \Lambda \cup \{\tau\}$ is a

transition labeling function. According to h, we can also partition T into k subsets corresponding to the behavior of different agents, i.e., $T = T_1 \cup T_2 \cup ... \cup T_k$, where transitions in T_i are labeled by actions from Λ_i with $i = 1, 2, ..., k$.

Then transitions $t_i \in T_i$, $t_j \in T_j$, where $i \neq j$ and $h(t_i) \neq \tau$, $h(t_j) \neq \tau$, are called *neighboring* if there exists a path in N connecting t_i and t_j, such that the other transitions along this path are labeled by τ, i.e., silent. Formally:

- $(t_i, t_j) \in F^*$, where F^* is the reflexive transitive closure of F and
- $\forall t \in T \setminus \{t_i, t_j\}$: if $(t_i, t) \in F^*$ and $(t, t_j) \in F^*$, then $\ell(t) = \tau$.

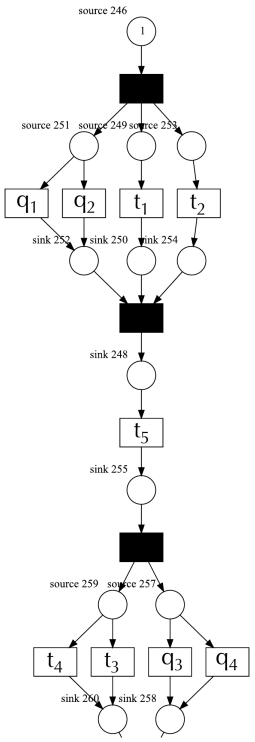
NbT(N) denotes the set of all neighboring transition pairs in N, where symmetric pairs of neighboring transitions are counted as a single pair, i.e., $(t_1, t_2) \in$ NbT(N) $\Leftrightarrow$ $(t_2, t_1) \notin$ NbT(N). Intuitively, the bigger the |NbT(N)| is, the less transparent and understandable the structure of N is with respect to agent interactions, since there are a lot of causally dependent transitions corresponding to the behavior of different agents. Further, we give an example of counting pairs of neighboring transitions.

Consider two LGWF-net fragments presented in Fig. 51. The first fragment (see Fig. 51a) is taken from the LGWF-net shown in Fig. 2. The second fragment (see Fig. 51b) is taken from the LGWF-net shown in Fig. 1.

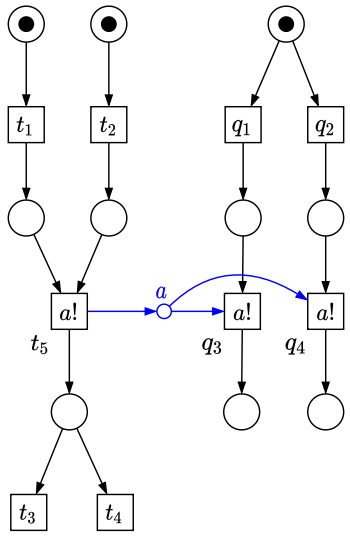
Recall that the LGWF-net shown in Fig. 2 has been discovered directly from an event log generated by the LGWF-net shown in Fig. 1. Both LGWF-nets perfectly fit this event log. However, as mentioned in Introduction, the direct discovery of a multi-agent system model may lead to inappropriate generalizations of agent behavior. For example, in the fragment shown in Fig. 51b, transition q_3 can fire only after transition q_1, while in the fragment shown in Fig. 51a, transition q_3 can fire after one of two transitions q_1 or q_2.

These LGWF-net fragments depict the behavior of a multi-agent system with two agents. The behavior of Agent 1 is represented by transitions $\{t_1, t_2, t_3, t_4, t_5\}$. The behavior of Agent 2 is represented by transitions $\{q_1, q_2, q_3, q_4\}$. The first

fragment, shown in Fig. 51a, has four pairs of neighboring transitions, i.e., $\{(q_1, t_5), (q_2, t_5), (t_5, q_3), (t_5, q_4)\}$. However, in the second fragment, shown in Fig. 51b, there are only two pairs of neighboring transitions, i.e., $\{(t_5, q_3), (t_5, q_4)\}$, corresponding exactly to the transitions through which Agents 1 and 2 interact.



(a) $|\mathrm{NbT(N)}| = 4$



(b) $|\mathrm{NbT(N)}| = 2$

Figure 51: Neighboring transitions

Further computation of neighboring transition pairs in the LGWF-nets, provided in Fig. 1 and Fig. 2, will lead to the following observations:

1. In a GWF-net discovered by Algorithm 1, the number of neighboring transitions directly corresponds to the interacting transitions.
2. In a GWF-net discovered directly from an event log of a multi-agent system, there are far more pairs of neighboring transitions, since transitions

corresponding to different agents are tightly connected.

5.5 Experimental Results

Table 8 presents the absolute values of the precision and simplicity of the reference, directly and compositionally discovered LGWF-nets of multi-agent systems, which architecture is described by interface patterns IP-1, ..., IP-12. In this table, we also provide the information on the artificial event logs, obtained by simulating the behavior of the reference LGWF-nets, including the total number of events together with the minimum, maximum, and average trace length in these event logs. The longest traces are represented in the event log of pattern IP-5, since there are parallel branches in the behavior of interacting agents. The most notable difference between the minimum and maximum trace lengths is represented in the event log of IP-7, since there are loops in the behavior of interacting agents.

For a better interpretation of the experimental results, Table 9 provides pair-wise comparison between the precision and simplicity evaluations given in Table 8. We have computed the percentage change in the characteristics of:

- directly discovered and reference LGWF-nets;
- compositionally discovered and reference LGWF-nets;
- compositionally and directly discovered LGWF-nets.

Based on these pair-wise comparison results, we report the main conclusions and outcomes from the experiments on evaluating the compositional process discovery approach below.

We start with the analysis of the simplicity comparisons given in Table 9.

An increase in the number of nodes in the directly and compositionally discovered LGWF-nets, next to the reference LGWF-nets, is mainly caused by additional τ-transitions. They connect the standard behavioral constructions such as the sequential, concurrent, or alternative control-flows of actions executed by different

agents. Conversely, the compositional process discovery shows an overall decrease or a moderate increase in the number of nodes compared with the direct process discovery since we separate the behavior of different agents. The separation of agent behavior is also justified by the changes in the number of neighboring transitions. One may observe a multiplicative increase in the number of the neighboring transitions in the directly discovered LGWF-nets. The compositionally discovered LGWF-nets have the same number of the neighboring transitions as the reference LGWF-nets. These transitions correspond exactly to actions through which agents interact, while the rest of agent behavior is independent since it is not involved in their interactions.

We next consider the precision comparisons also provided in Table 9.

Most directly discovered LGWF-nets improve the precision since they are far more oriented to the corresponding event logs. The precision of the reference and compositionally discovered LGWF-nets are lower next to the directly discovered LGWF-nets since the separation of agent behavior leads to a corresponding increase in the amount of unseen behavior, as shown in Fig. 50. This precision decrease is a payment for making process models of multi-agent systems architecture-aware. However, in the case of the interface patterns with complicated and mixed agent interactions, namely IP-7, IP-9, ..., and IP-12, we observe a decrease or a negligible increase in precision. The inappropriate generalizations of agent behavior were the main reason for this precision decrease. In conclusion to the precision comparative analysis, we also see that the compositionally discovered GWF-nets preserve almost the same precision level next to the reference LGWF-nets since changes in the corresponding values are less than 10%.

To sum up the discussion of the experimental results, we take a closer look at the outcomes reported for patterns IP-2 and IP-7. Following the steps of our experiment plan for these interface patterns, we encountered the following issues:

- interface pattern inconsistencies (IP-2);
- the most notable decrease in the precision (IP-7).

Table 8: Experimental results: absolute values

	IP-1	**IP-2**	**IP-3**	**IP-4**	**IP-5**	**IP-6**	**IP-7**	**IP-8**	**IP-9**	**IP-10**	**IP-11**	**IP-12**
						Event logs						
Events	95052	149988	92668	102404	182452	123322	88068	157098	115000	102548	160000	88089
MIN trace	17	29	17	18	36	24	8	30	23	20	32	17
AVG trace	19	30	19	20	36	25	18	31	23	21	32	18
MAX trace	21	31	20	23	37	25	29	32	23	21	32	18
						Reference LGWF-nets used for event log generation						
Places	35	52	45	41	59	60	31	70	53	41	50	44
Transitions	31	48	43	35	50	53	30	58	52	37	44	42
Arcs	73	110	96	86	121	131	78	148	128	89	103	97
NbT	4	4	2	6	7	7	12	8	6	6	6	3
Precision	0.73349	0.47346	0.77810	0.79798	0.37020	0.60955	0.82935	0.54382	0.76541	0.83729	0.43610	0.77731
						LGWF-nets discovered directly from event logs						
Places	52	74	83	56	88	97	30	107	68	49	84	73
Transitions	51	69	71	55	78	81	48	84	79	53	80	67
Arcs	126	182	178	134	210	214	104	230	186	126	202	168
NbT	48	124	46	48	81	74	69	117	75	47	71	41
Precision	0.75230	0.57704	0.89348	0.75958	0.48480	0.76818	0.32359	0.66462	0.60369	0.69177	0.45910	0.80212
						Compositional process discovery						
Places	46	72	54	57	94	81	39	96	66	49	69	55
Transitions	41	71	51	49	80	71	38	78	64	45	63	52
Arcs	96	176	114	119	211	177	92	200	155	109	155	121
NbT	4	4	2	6	7	7	12	8	6	6	6	3
Precision	0.73639	0.43663	0.78777	0.80728	0.40350	0.62615	0.82180	0.56989	0.76785	0.81475	0.46640	0.77522

Table 9: Experimental results: changes in simplicity and precision evaluations

	IP-1	IP-2	IP-3	IP-4	IP-5	IP-6	IP-7	IP-8	IP-9	IP-10	IP-11	IP-12
Directly discovered LGWF-nets compared to reference LGWF-nets												
Places	+49%	+42%	+84%	+37%	+49%	+62%	-3%	+53%	+28%	+20%	+68%	+66%
Transitions	+65%	+44%	+65%	+57%	+56%	+53%	+60%	+45%	+52%	+43%	+82%	+60%
Arcs	+72%	+66%	+85%	+56%	+74%	+63%	+33%	+55%	+45%	+42%	+96%	+73%
NbT	×12	×31	×23	×12	×11.6	×10.6	×5.6	×14.6	×12.5	×7.8	×11.8	×13.7
Precision	+3%	+22%	+15%	-5%	+31%	+26%	-61%	+22%	-21%	-17%	+5%	+3%
Compositionally discovered LGWF-nets compared to reference LGWF-nets												
Places	+31%	+40%	+20%	+39%	+59%	+35%	+26%	+37%	+25%	+20%	+38%	+25%
Transitions	+32%	+48%	+19%	+40%	+60%	+34%	+27%	+35%	+23%	+22%	+43%	+24%
Arcs	+32%	+60%	+19%	+38%	+74%	+35%	+18%	+35%	+21%	+23%	+51%	+25%
NbT	coincides with the values of reference GWF-nets											
Precision	+0,4%	-8%	+1%	+1%	+9%	+3%	-1%	+5%	+0.3%	-3%	+7%	-0.3%
Compositionally discovered LGWF-nets compared to directly discovered LGWF-nets												
Places	-12%	-3%	-35%	+2%	+7%	-17%	+30%	-10%	-3%	0%	-18%	-25%
Transitions	-20%	+3%	-28%	-11%	+3%	-12%	-21%	-7%	-19%	-15%	-21%	-22%
Arcs	-24%	-3%	-36%	-11%	+1%	-17%	-12%	-13%	-17%	-14%	-23%	-28%
NbT	-92%	-97%	-96%	-88%	-91%	-91%	-83%	-94%	-92%	-87%	-92%	-93%
Precision	-2%	-24%	-12%	+6%	-17%	-19%	×2.5	-14%	+27%	+18%	+2%	-3%

We further discuss the reasons for these problems.

5.5.1 Pattern Inconsistencies: the Case of IP-2

The experiment with the asynchronous interface pattern IP-2 shown in Fig. 39b led to the following problem. Agent LGWF-nets, N_1 and N_2, discovered from log projections, were not the proper refinements of A_1 and A_2 in IP-2, according to the requirement of Definition 16. Thus, the corresponding isRefinement test in Algorithm 1 was not passed.

As mentioned in Section 5.1, in the detailed description of the experiment plan, we would try to reconfigure an interface pattern in this case. Then, we determined that there exist two sequences of refinement transformations (see Section 3.4) that lead from A'_1 and A'_2 shown in Fig. 52 to agent GWF-nets N_1 and N_2.

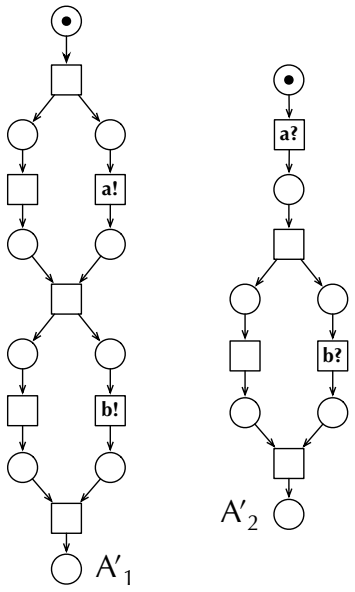


Figure 52: Modifications of A_1 and A_2 in IP-2

Intuitively, it can be seen that the two pairs of concurrent actions, "a!", "b!" and "a?", "b?", were discovered as sequential actions respectively. The main reason

for this is the lack of different process executions in an event log generated by the reference LGWF-net.

Having considered $A_1' \circledast A_2'$ as the new interface pattern and verified its soundness (see Proposition 13), we actually experimented with the modified version of the interface pattern IP-2.

5.5.2 Precision Drop: the Case of IP-7

The experiment with the multiple transmission interface pattern IP-7, shown in Fig. 39g, also deserves to be highlighted. The directly discovered LGWF-net exhibits a sharp decrease in its precision compared to the reference and compositionally discovered LGWF-nets.

Figure 53 shows the LGWF-net discovered directly from an event log generated by the reference LGWF-net of pattern IP-7. As seen from this LGWF-net, its structure contains several joint blocks of actions executed by different agents. The structure of this model does not cover the interaction-oriented architecture viewpoints of a system with two interacting agents exchanging messages until one of them decides to stop the exchange.

Thus, the complicated nature of agent interactions can hardly be reconstructed directly from an event log of a multi-agent system. The identification of agent behavior and the interface pattern refinement check allows us to decompose this problem and improve the quality of a multi-agent system model.

5.6 Technical Support of Experiments

According to the main plan discussed in Section 5.1, experiments on discovering architecture-aware LGWF-nets from event logs of multi-agent system have been conducted using a PC with the following characteristics:

1. CPU Intel Core i7 3,70GHz;

2. 32 Gb RAM;

3. 64-bit OS Windows 10 Pro.

The generation and filtration of artificial event logs, discovery of agent LGWF-nets, and the precision evaluation have been supported by the *ProM* software [91]. This is the plugin-extendable tool, where various process discovery algorithms have been implemented.

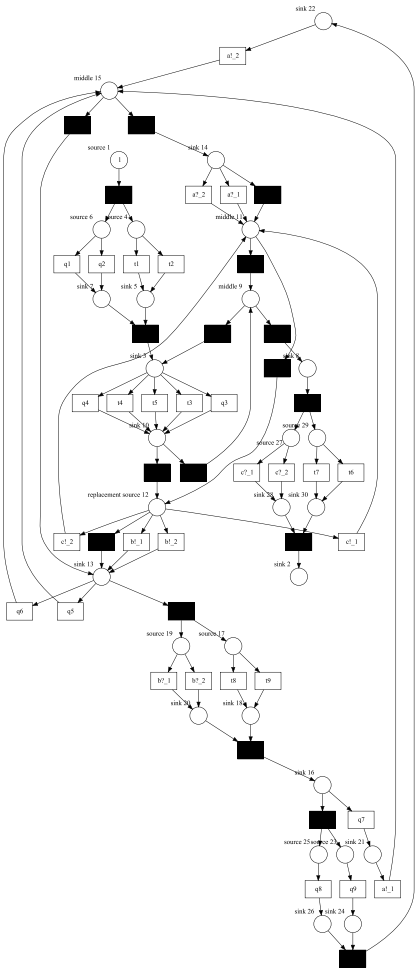


Figure 53: Directly discovered LGWF-net: interface pattern IP-7

The experimental data, including the artificial event logs, reference, directly, and compositionally discovered GWF-nets, have been published in the open Zenodo repository [92].

5.7 Conclusions of Chapter 5

This chapter discussed the outcomes from the series of experiments conducted to evaluate the compositional process discovery algorithm. We compared the quality of the process models discovered directly from the artificial event logs of multi-agent systems with the quality of the process models discovered using the interface patterns with respect to the central hypothesis of the compositional approach. The experimental results confirm the overall improvement in the structure of architecture-aware process models of multi-agent systems since agent behavior, not involved in their interactions, is structurally separated.

However, the experimental results found that the agent interaction requirements imposed by interface patterns might not be fully satisfied by agent GWF-nets discovered from filtered sub-logs. The main reason for this problem is the incompleteness of event logs. They represent only a "finite snapshot" of all possible executions generated by concurrent interactions among agents in multi-agent systems. To tackle the problem, one should either correspondingly adapt an interface pattern verifying its soundness, as exemplified in Section 5.5.1, or process event logs with a bigger number of different trace classes.

Another limitation is the manual selection of an interface pattern and the manual construction of refinement transformation sequences using Definition 16. Manual work can result in the wrong pattern choice and mistakes in checking whether an agent GWF-net is a refinement of the respective subnet in an interface pattern. We plan to overcome these issues by developing an algorithm for the refinement check isRefinement (see Algorithm 1) and by extending the collection of typical and sound interface patterns.

Conclusions and Future Work

This dissertation proposed the algorithm for the compositional discovery of process models from event logs of multi-agent systems. This algorihm is based on composing process models representing the individual agent behavior synthesized from a set of filtered sub-logs. The composition is regulated by an interface pattern that describes agent interactions at the highly abstract level. If all agent models can be mapped on the respective parts in the interface pattern, we guarantee that:

1. The discovered process model of a multi-agent system perfectly fits an initial event log (see Corollary 5).
2. The discovered process model of a multi-agent system inherits the soundness of individual agents an of the interface pattern (see Corollary 6).

The correctness of the proposed algorithm for the compositional discovery of process models from event logs of multi-agent systems is based on the following three aspects:

1. Theoretical backgrounds of representing and filtering event logs and of the asynchronous-synchronous composition of labeled generalized workflow nets (LGWF-nets).
2. The technique, based on α-morphisms and structural transformations, for mapping agent models on the corresponding subnets in the interface pattern.

3. The collection of sound interface patterns that describe typical agent interactions and preserve their soundness.

The proposed approach is applicable to multi-agent systems with components that can be represented as business processes. The control-flow of these processes can be formalized using generalized workflow nets, making the soundness property relevant for the analysis. The main limitation of the compositional process discovery is the manual selection of interface patterns according to information provided by experts. This can result in the further adaptation and soundness verification of modified interface patterns. However, the number of interacting actions is usually significantly less than the number of local actions of agents.

As for future research, we plan to continue our work in several directions that are also focused on overcoming the limitations. Firstly, the application of α-morphisms and the corresponding structural transformations does not allow refining acyclic interface patterns with cyclic behavior. We want to consider possible constraint relaxations, such that the overall correctness is preserved. Then the applicability of interface patterns will be extended. Secondly, we plan to augment the presented collection of interface patterns with new interaction models, especially considering the broadcast communication, and apply the compositional approach to real-life examples of event logs. Finally, we also plan to work on an approach to identifying interfaces from event logs of multi-agent systems.

Acknowledgements

In the following lines, I want to express my gratitude to people who helped me during the four years of work on the dissertation.

Firstly, I am grateful to my supervisors, *Irina Lomazova* and *Lucia Pomello*, for their support and patience, for hours of fruitful discussions, for their constructive criticism, for believing in the "mathematical" side of me.

My enormous thanks are also to *Luca Bernardinello*, *Andiran Puerto*, and *Federica Adobbati* from the "Models of Concurrency, Computation and Communication (MC3)" research group at the University of Milano-Bicocca, where I spent almost five wonderful months working on many different theoretical aspects. You always questioned my ideas — this taught me how to build a solid foundation for my reasoning.

Laboratory for Process-Aware Information Systems (PAIS) at the HSE University has pride of place in my scientific life. I thank *Alex Mitsyuk* and *Sergey Shershakov* for their active participation at Lab meetings where we discussed intermediate results.

Ilona, *Ksenia*, *Stasya*, and *Tatyana Vasilievna* are exactly those people in the Faculty of Computer Science at HSE to whom I can always turn for help in any matter. Thank you for being here.

Last words on this page are devoted to the two most important *women* in my life, *Svetlana* and *Valentina*, to my mom and to my granny. I know that you will support me in all my endeavors and aspirations, no matter how crazy and reckless the can sometimes be.

References

[1] W. van der Aalst, *Process Mining: Data Science in Action*. Springer, Heidelberg, 2016.

[2] A. Augusto, R. Conforti, M. Dumas, M. Rosa, F. Maggi, A. Marrella, M. Mecella, and A. Soo, "Automated discovery of process models from event logs: Review and benchmark," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 4, pp. 686–705, 2019.

[3] W. Reisig, *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer, Heidelberg, 2013.

[4] A. Kalenkova, W. van der Aalst, I. Lomaozva, and V. Rubin, "Process mining using bpmn: relating event logs and process models," *Software and Systems Modeling*, vol. 16, pp. 1019–1048, 2017.

[5] J. Carmona, B. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking: Relating Processes and Models*. Springer Heidelberg, 2018.

[6] S. Leemans, D. Fahland, and W. van der Aalst, "Discovering block-structured process models from event logs - a constructive approach," in *Application and Theory of Petri Nets and Concurrency* (J. Colom and J. Desel, eds.), vol. 7927 of *Lecture Notes in Computer Science*, pp. 311–329, Springer Heidelberg, 2013.

[7] W. van der Aalst, "Workflow verification: Finding control-flow errors using petri-net-based techniques," in *Business Process Management: Models, Tech-*

niques, and Empirical Studies, vol. 1806 of *Lecture Notes in Computer Science*, pp. 161–183, Springer Heidelberg, 2000.

[8] W. van der Aalst, K. van Hee, A. ter Hofstede, N. Sidorova, H. Verbeek, M. Voorhoeve, and M. Wynn, "Soundness of workflow nets: classification, decidability, and analysis," *Formal Aspects of Computing*, vol. 23, no. 3, pp. 333–363, 2011.

[9] A. Barros, M. Dumas, and A. ter Hofstede, "Service interaction patterns," in *Business Process Management*, vol. 3649 of *Lecture Notes in Computer Science*, pp. 302–318, Springer Heidelberg, 2005.

[10] W. van der Aalst, "Workflow verification: Finding control-flow errors using petri-net-based techniques," in *Business Process Management: Models, Techniques, and Empirical Studies* (W. van der Aalst, J. Desel, and A. Oberweis, eds.), vol. 1806 of *Lecture Notes in Computer Science*, pp. 161–183, Springer, Heidelberg, 2000.

[11] G. Rozenberg and J. Engelfriet, "Elementary net systems," in *Lectures on Petri Nets I: Basic Models*, vol. 1491 of *LNCS*, pp. 12–121, Springer, Heidelberg, 1998.

[12] J. Esparza, S. Römer, and W. Vogler, "An improvement of mcmillan's unfolding algorithm," *Formal Methods in System Design*, vol. 20, pp. 285–310, 2002.

[13] J. Esparza, S. R$\acute{}$omer, and W. Vogler, "An improvement of mcmillan's unfolding algorithm," *Formal Methods in System Design*, vol. 20, pp. 285–310, 2002.

[14] S. Mac Lane, *Categories for the Working Mathematician*. Springer, 1978.

[15] L. Bernardinello, E. Mangioni, and L. Pomello, "Local state refinement and composition of elementary net systems: an approach based on morphisms," in *Transaction on Petri Nets and Other Models of Concurrency VIII*, vol. 8100 of *Lecture Notes in Computer Science*, pp. 48–70, Springer, Heidelberg, 2013.

[16] V. E. Kotov, "An algebra for parallelism based on petri nets," in *Mathematical Foundations of Computer Science 1978* (J. Winkowski, ed.), vol. 64 of *Lecture Notes in Computer Science*, pp. 39–55, Springer, Heidelberg, 1978.

[17] E. Best, R. Devillers, and J. Hall, "The box calculus: A new causal algebra with multi-label communication," in *Advances in Petri Nets 1992*, vol. 609 of *Lecture Notes in Computer Science*, pp. 21–69, Springer, 1992.

[18] C. Girault and R. Valk, *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications*. Springer, Heidelberg, 2003.

[19] W. Reisig, "Associative composition of components with double-sided interfaces," *Acta Informatica*, vol. 56, pp. 229–253, 2019.

[20] W. Reisig, "Composition of component models – a key to construct big systems," in *Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles*, vol. 12477 of *Lecture Notes in Computer Science*, pp. 171–188, Springer International Publishing, 2020.

[21] P. Fettke and W. Reisig, "Modelling service-oriented systems and cloud services with heraklit," in *Advances in Service-Oriented and Cloud Computing*, vol. 1360 of *Communications in Computer and Information Science*, (Cham), pp. 77–89, Springer International Publishing, 2021.

[22] P. Baldan, A. Corradini, H. Ehrig, and R. Heckel, "Compositional modeling of reactive systems using open nets," in *International Conference on Concurrency Theory. CONCUR 2001*, vol. 2154 of *Lecture Notes in Computer Science*, pp. 502–518, Springer, Heidelberg, 2001.

[23] K. van Hee, N. Sidorova, and J. van der Werf, "Construction of asynchronous communicating systems: Weak termination guaranteed!," in *Software Composition*, vol. 6144 of *LNCS*, pp. 106–121, Springer, Heidelberg, 2010.

[24] K. M. van Hee, A. J. Mooij, N. Sidorova, and J. van der Werf, "Soundness-preserving refinements of service compositions," in *Web Services and Formal Methods*, vol. 6551 of *LNCS*, pp. 131–145, Springer, Heidelberg, 2011.

[25] F. De Cindo, G. De Michelis, L. Pomello, and C. Simone, "Superposed automata nets," in *Application and Theory of Petri Nets*, vol. 52 of *Informatik-Fachberichte*, pp. 269–279, Springer, Heidelberg, 1982.

[26] S. Haddad, R. Hennicker, and M. Møller, "Channel properties of asynchronously composed petri nets," in *Application and Theory of Petri Nets and Concurrency. PETRI NETS 2013*, vol. 7927 of *Lecture Notes in Computer Science*, pp. 369–388, Springer, Heidelberg, 2013.

[27] Y. Souissi and G. Memmi, "Composition of nets via a communication medium," in *Advances in Petri Nets 1990* (G. Rozenberg, ed.), vol. 483 of *Lecture Notes in Computer Science*, pp. 457–470, Springer, Heidelberg, 1991.

[28] Y. Souissi, "On liveness preservation by composition of nets via a set of places," in *Advances in Petri Nets 1991* (G. Rozenberg, ed.), vol. 524 of *Lecture Notes in Computer Science*, pp. 277–295, Springer, Heidelberg, 1991.

[29] C. Stahl and K. Wolf, "Deciding service composition and substitutability using extended operating guidelines," *Data & Knowledge Engineering*, vol. 68, pp. 819–833, 2009.

[30] G. Winskel, "Petri nets, algebras, morphisms, and compositionality," *Information and Computation*, vol. 72, no. 3, pp. 197–238, 2007.

[31] J. Meseguer and U. Montanari, "Petri nets are monoids," *Information and Computation*, vol. 88, no. 2, pp. 105–155, 1990.

[32] M. Nielsen, G. Rozenberg, and P. Thiagarajan, "Elementary transition systems," *Theoretical Computer Science*, vol. 96, no. 1, pp. 3–33, 1992.

[33] M. A. Bednarczyk, L. Bernardinello, B. Caillaud, W. Pawłowski, and L. Pomello, "Modular system development with pullbacks," in *Applications and Theory of Petri Nets 2003. ICATPN 2003*, vol. 2679 of *Lecture Notes in Computer Science*, pp. 140–160, Springer, Heidelberg, 2003.

[34] J. Padberg and M. Urbášek, "Rule-based refinement of petri nets: A survey," in *Petri Net Technology for Communication-Based Systems: Advances in Petri Nets* (H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, eds.), vol. 2427 of *Lecture Notes in Computer Science*, pp. 161–196, Springer, Heidelberg, 2003.

[35] E. Fabre, "On the construction of pullbacks for safe petri nets," in *Petri Nets and Other Models of Concurrency - ICATPN 2006*, vol. 4024 of *Lecture Notes in Computer Science*, pp. 166–180, Springer, Heidelberg, 2006.

[36] L. Bernardinello, E. Monticelli, and L. Pomello, "On preserving structural and behavioral properties by composing net systems on interfaces," *Fundamenta Informaticae*, vol. 80, no. 1–3, pp. 31–47, 2007.

[37] J. Desel and A. Merceron, "Vicinity respecting homomorphisms for abstracting system requirements," in *Transactions on Petri Nets and Other Models of Concurrency IV* (K. Jensen, S. Donatelli, and M. Koutny, eds.), vol. 6550 of *Lecture Notes in Computer Science*, pp. 1–20, Springer, Heidelberg, 2010.

[38] J. Siegeris and A. Zimmermann, "Workflow model compositions preserving relaxed soundness," in *BPM 2006*, vol. 4102 of *LNCS*, pp. 177–192, Springer, 2006.

[39] I. A. Lomazova and I. V. Romanov, "Analyzing compatibility of services via resource conformance," *Fundamenta Informaticae*, vol. 128, no. 1–2, pp. 129–141, 2013.

[40] I. A. Lomazova, "Interacting workflow nets for workflow process re-engineering," *Fundamenta Informaticae*, vol. 101, no. 1–2, pp. 59–70, 2010.

[41] Y. Cardinale, J. El Haddad, M. Manouvrier, and M. Rukoz, "Web service composition based on petri nets: Review and contribution," in *Resource Discovery* (Z. Lacroix, E. Ruckhaus, and M.-E. Vidal, eds.), vol. 8194 of *Lecture Notes in Computer Science*, pp. 83–122, Springer, Heidelberg, 2013.

[42] V. Pankratius and W. Stucky, "A formal foundation for workflow composition, workflow view definition, and workflow normalization based on petri nets," in *Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005)*, vol. 43 of *CRPIT*, (Newcastle, Australia), pp. 79–88, ACS, 2005.

[43] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[44] G. Berthelot, "Checking properties of nets using transformations," in *Advances in Petri Nets 1985* (G. Rozenberg, ed.), vol. 222 of *Lecture Notes in Computer Science*, pp. 19–40, Springer, Heidelberg, 1986.

[45] G. Berthelot and G. Roucairol, "Reduction of petri-nets," in *Mathematical Foundations of Computer Science 1976* (A. Mazurkiewicz, ed.), vol. 45 of *Lecture Notes in Computer Science*, pp. 202–209, Springer, Heidelberg, 1976.

[46] T. Murata and I. Suzuki, "A method for stepwise refinement and abstraction of petri nets," *Journal of Computer and System Sciences*, vol. 27, no. 1, pp. 51–76, 1983.

[47] R. Valette, "Analysis of Petri nets by stepwise refinements," *Journal of Computer and System Sciences*, vol. 18, no. 1, pp. 35–46, 1979.

[48] M. Hack, "Analysis of production schemata by Petri nets. TR-94," MIT, Boston, 1972.

[49] J. Desel and J. Esparza, *Free Choice Petri Nets*. Cambridge University Press, 1995.

[50] J. Esparza and M. Silva, "Top-down synthesis of live and bounded free choice nets," in *Advances in Petri Nets 1991* (G. Rozenberg, ed.), vol. 524 of *Lecture Notes in Computer Science*, pp. 118–139, Springer, Heidelberg, 1991.

[51] H. Genrich and P. Thiagarajan, "A theory of bipolar synchronisation schemes," *Theoretical Computer Science*, vol. 30, no. 3, pp. 241–318, 1984.

[52] H. Ehrig, K. Hoffman, and J. Padberg, "Transformations of petri nets," *Electronic Notes in Computer Science*, vol. 148, no. 1, pp. 151–172, 2006.

[53] P. Schnoebelen and N. Sidorova, "Bisimulation and the reduction of Petri nets," in *Application and Theory of Petri Nets 2000* (M. Nielsen and D. Simpson, eds.), vol. 1825 of *Lecture Notes in Computer Science*, pp. 409–423, Springer, Heidelberg, 2000.

[54] I. A. Lomazova, "Resource equivalences in Petri nets," in *Application and Theory of Petri Nets and Concurrency* (W. van der Aalst and E. Best, eds.), vol. 10258 of *Lecture Notes in Computer Science*, pp. 19–34, Springer, Heidelberg, 2017.

[55] M. Nielsen and G. Winskel, "Petri nets and bisimulations," *Theoretical Computer Science*, vol. 153, pp. 211–244, 1996.

[56] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[57] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, pp. 5–51, 2003.

[58] L. de Alfaro and T. Henzinger, "Interface-based design," in *Engineering Theories of Software Intensive Systems*, pp. 83–104, Springer Netherlands, 2005.

[59] C. Günther and W. van der Aalst, "Fuzzy mining – adaptive process simplification based on multi-perspective metrics," in *Business Process Management*

(G. Alonso, P. Dadam, and M. Rosemann, eds.), vol. 4714 of *Lecture Notes in Computer Science*, pp. 328–343, Springer Heidelberg, 2007.

[60] A. Weijters and J. Ribeiro, "Flexible heuristics miner (fhm)," in *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pp. 310–317, 2011.

[61] J. van der Werf, B. van Dongen, C. Hurkens, and A. Serebrenik, "Process discovery using integer linear programming," in *Applications and Theory of Petri Nets* (K. van Hee and R. Valk, eds.), vol. 5062 of *Lecture Notes in Computer Science*, pp. 368–387, Springer Heidelberg, 2008.

[62] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser, "Process mining based on regions of languages," in *Business Process Management* (G. Alonso, P. Dadam, and M. Rosemann, eds.), vol. 4714 of *Lecture Notes in Computer Science*, (Berlin, Heidelberg), pp. 375–383, Springer Heidelberg, 2007.

[63] W. van der Aalst, A. de Medeiros, and A. Weijters, "Genetic process mining," in *Applications and Theory of Petri Nets 2005* (G. Ciardo and P. Darondeau, eds.), vol. 3536 of *Lecture Notes in Computer Science*, pp. 48–69, Springer Heidelberg, 2005.

[64] E. Badouel, L. Bernardinello, and P. Darondeau, *Petri Net Synthesis*. Springer, Heidelberg, 2015.

[65] E. Badouel and P. Darondeau, "Theory of regions," in *Lectures on Petri Nets I: Basic Models: Advances in Petri Nets*, vol. 1491 of *Lecture Notes in Computer Science*, pp. 529–586, Springer Heidelberg, 1998.

[66] J. Buijs, B. van Dongen, and W. van der Aalst, "On the role of fitness, precision, generalization and simplicity in process discovery," in *On the Move to Meaningful Internet Systems: OTM 2012*, vol. 7565 of *Lecture Notes in Computer Science*, pp. 305–322, Springer Heidelberg, 2012.

[67] W. van der Aalst, "Relating process models and event logs – 21 conformance propositions," in *Proceedings of the International Workshop on Algorithms and Theories for the Analysis of Event Data 2018*, vol. 2115 of *CEUR Workshop Proceedings*, pp. 56–74, CEUR-WS.org, 2018.

[68] W. van der Aalst and C. Gunther, "Finding structure in unstructured processes: The case for process mining," in *Seventh International Conference on Application of Concurrency to System Design (ACSD 2007)*, pp. 3–12, 2007.

[69] J. Buijs, *Flexible evolutionary algorithms for mining structured process models*. PhD thesis, Eindhoven University of Technology, 2014.

[70] J. De Smedt, J. De Weerdt, and J. Vanthienen, "Multi-paradigm process mining: Retrieving better models by combining rules and sequences," in *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*, vol. 8841 of *Lecture Notes in Computer Science*, pp. 446–453, Springer Heidelberg, 2014.

[71] J. de San Pedro and J. Cortadella, "Mining structured petri nets for the visualization of process behavior," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, p. 839–846, ACM, 2016.

[72] A. Kalenkova, A. Lomazova, and W. van der Aalst, "Process model discovery: A method based on transition system decomposition," in *Application and Theory of Petri Nets and Concurrency* (G. Ciardo and E. Kindler, eds.), vol. 8489 of *Lecture Notes in Computer Science*, pp. 71–90, Springer Heidelberg, 2014.

[73] A. Kalenkova and I. Lomaozva, "Discovery of cancellation regions within process mining techniques," *Fundamenta Informaticae*, vol. 133, pp. 197–209, 2014.

[74] W. van der Aalst, A. Kalenkova, V. Rubin, and E. Verbeek, "Process discovery using localized events," in *Application and Theory of Petri Nets and Concurrency*

(R. Devillers and A. Valmari, eds.), vol. 9115 of *Lecture Notes in Computer Science*, pp. 287–308, Springer Heidelberg, 2015.

[75] W. van der Aalst and A. Berti, "Discovering object-centric petri nets," *Fundamenta Informaticae*, vol. 175, pp. 1–40, 2020.

[76] M. Stierle, S. Zilke, S. Dunzer, J. Tenscher, and G. Karagegova, "Design principles for comprehensible process discovery in process mining," in *ECIS 2020 Proceedings. Research Papers*, AIS eLibrary, 2020.

[77] G. Decker and A. Barros, "Interaction modeling using bpmn," in *Business Process Management Workshops* (A. ter Hofstede, B. Benatallah, and H.-Y. Paik, eds.), vol. 4928 of *Lecture Notes in Computer Science*, pp. 208–219, Springer Heidelberg, 2008.

[78] D. Campagna, C. Kavka, and L. Onesti, "Bpmn 2.0 and the service interaction patterns: Can we support them all?," in *Software Technologies*, vol. 555 of *Communications in Computer and Information Science*, pp. 3–20, Springer Heidelberg, 2015.

[79] G. Decker, F. Puhlmann, and M. Weske, "Formalizing service interactions," in *Business Process Management* (S. Dustdar, J. L. Fiadeiro, and A. P. Sheth, eds.), vol. 4102 of *Lecture Notes in Computer Science*, pp. 414–419, Springer Heidelberg, 2006.

[80] W. van der Aalst, A. Mooij, C. Stahl, and K. Wolf, "Service interaction: Patterns, formalization, and analysis," in *SFM 2009: Formal Methods for Web Services* (M. Bernardo, L. Padovani, and G. Zavattaro, eds.), vol. 5569 of *Lecture Notes in Computer Science*, pp. 42–88, Springer Heidelberg, 2009.

[81] A. Burattin, "Multiperspective process randomization with online and offline simulations," in *BPMD 2016*, vol. 1789 of *CEUR Workshop Proceedings*, pp. 1–6, CEUR-WS.org, 2016.

[82] Z. Yan, R. Dijkman, and P. Grefen, "Generating process model collections," *Software & Systems Modeling*, vol. 16, pp. 979–995, 2017.

[83] T. Jouck and B. Depaire, "PTandLogGenerator: A generator for artificial event data," in *BPMD 2016*, vol. 1789 of *CEUR Workshop Proceedings*, pp. 23–27, CEUR-WS.org, 2016.

[84] K. van Hee and Z. Liu, "Generating benchmarks by random stepwise refinement of petri nets," in *ACSD 2010*, vol. 827 of *CEUR Workshop Proceedings*, pp. 403–417, CEUR-WS.org, 2010.

[85] I. Shugurov and A. Mitsyuk, "Generation of a set of event logs with noise," in *Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE 2014)*, pp. 88–95, 2014.

[86] I. Shugurov and A. Mitsyuk, "Generation of a set of event logs with noise," in *Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE 2014)*, pp. 88–95, 2014.

[87] A. Mitsyuk, I. Shugurov, A. Kalenkova, and W. van der Aalst, "Generating event logs for high-level process models," *Simulation Modeling Practice and Theory*, vol. 74, pp. 1–16, 2017.

[88] A. de Medeiros and C. Günther, "Process mining: Using cpn tools to create test logs form mining algorithms," in *Proceedings of CPN 2005*, vol. 576 of *DAIMI Report Series*, pp. 177–190, 2005.

[89] C. Di Ciccio, M. L. Bernardi, M. Cimitile, and F. M. Maggi, "Generating event logs through the simulation of declare models," in *Enterprise and Organizational Modeling and Simulation* (J. Barjis, R. Pergl, and E. Babkin, eds.), vol. 231 of *Lecture Notes in Business Information Processing*, pp. 20–36, Springer Heidelberg, 2015.

[90] A. Adriansyah, *Aligning observed and modeled behavior*. PhD thesis, Technische Universiteit Eindhoven, 2014.

[91] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst, "The ProM framework: A new era in process mining tool support," in *Applications and Theory of Petri Nets 2005* (G. Ciardo and P. Darondeau, eds.), pp. 444–454, Springer, Heidelberg, 2005.

[92] R. Nesterov, "Compositional discovery of architecture-aware and sound process models from event logs of multi-agent systems: experimental data. (Version 1) [Data set]," May 2021, `https://doi.org/10.5281/zenodo.5830863`.