Verification of soundness and other properties of business processes

Verification of Soundness and Other Properties of Business Processes

Olivia Oanea

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Oanea, Olivia Ionela

Verification of soundness and other properties of business processes / door Olivia Ionela Oanea.
Eindhoven : Technische Universiteit Eindhoven, 2007.
Proefschrift. ISBN 978-90-386-11662
NUR 993
Subject headings: Petri nets / software verification / programming / formal methods
CR Subject Classification (1998): D.2.2, D.2.4, H.4.1, D.3.1, F.1.2, F.3.2, F.4.1, D.2.5, D.2.11
Keywords: Verification / Modeling / Business processes / Petri nets / Workflow / Soundness

The work reported in this thesis has been carried out under the auspices of Beta Research School for Operations Management and Logistics.

This research was supported by the NWO Open Competition project MoVeBP, project number 612.000.315

Beta Dissertation Series D101

Printed by University Press Facilities, Eindhoven

Verification of Soundness and Other Properties of Business Processes

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op dinsdag 11 december 2007 om 14.00 uur

door

Olivia Ionela Oanea

geboren te Iași, Roemenië

Dit proefschrift is goedgekeurd door de promotor:

prof.dr. K.M. van Hee

Copromotor:

dr. N. Sidorova

Table of Contents

1

Introduction

1.1 Motivation and Background

Business process modeling

Business processes are nowadays recognized as cornerstones of successful organizations. Business processes modeling has emerged due to the need of documenting, communication and knowledge transfer regarding processes in the business world. Nowadays, companies put more stress on the evaluation, verification, optimization and redesign of their processes in order to keep companies at a competitive level.

In the last two decades a great need for automation of business processes has risen. In the past, every aspect of business was handled manually and all the knowledge transfer necessary for supporting the continuity of business processes and information exchange was done through written or oral communication. Due to the emergence of ICT and the globalization of enterprises, more effort has been invested in automating almost every aspect of business that can be automated, to reduce the time and the costs of human resources. A typical example are travel agencies, where internet booking has removed human agents handling the booking. In parallel to this, the need to formalize business processes has arisen in order to incorporate them into the applications running the business.

A *business process* is a set of coordinated tasks or activities, conducted by both people and other resources, that leads to accomplishing a specific organizational goal, typically a product or a service.

During the nineties, the focus of Business Process Management has been on the development of a new type of software components for the enactment of business processes, so-called WorkFlow Management Systems (WFMSs). WFMSs are configured by process models; therefore many process modeling tools have interfaces with workflow management systems in order to develop, execute and exchange these models. WFMSs can be used stand alone (e.g. Staffware Process Suite, Filenet, FLOWer, WebSphere MQ Workflow), but mostly they are integrated with other software components like document management systems, (e.g. BizTalk). Most enterprise information systems (e.g. ERP-systems) have nowadays a workflow engine as a component as well. For instance, SAP WebFlow is the workflow component of SAP offering all the functionality typically present in custom workflow products. WFMSs have appeared in other application domains, like healthcare creating, but no breakthrough has happened yet.

Many ad-hoc languages for specification of business processes have been proposed, e.g. event-driven process chains (EPCs) [41] or flowcharts [108]. The important vendors of software such as IBM and Microsoft joined forces to develop standard languages to exchange process models (WSBPEL [17], BPMN, UML activity diagrams [143, 48]). The semantics of business process modeling languages is mostly often given by a simulator as a part of a modeling tool (like it is given by compilers for programming languages). At the same time,

there exist many academic process modeling languages that do have formal semantics but they are less appropriate for being used in concrete industrial application domains.

One of the most well-established process modeling languages in academia is Petri nets. Petri nets [110, 118] are a well-known formalism for concurrent processes, with an intuitive graphical representation which makes them suitable for the use by both verification specialists and domain experts. Several extensions of Petri nets have been proposed to enhance the modeling power of Petri nets. One extension is the addition of color [74], which allow to manipulate data in diverse forms. Special classes of (high level) Petri nets, e.g. hierarchical colored Petri nets [74] and modular Petri nets [30], have been proposed to allow compositional modeling of Petri nets. To reflect the object-oriented paradigm, new Petri net classes have been developed: classes of object oriented Petri nets [85, 24], "nets in nets" [136], nested nets [93], etc.

In this thesis, we study business process models taking Petri nets as an underlying concept.

Verification of business processes

As business processes have become more complex, the probability of making errors in their design has increased. Errors in business processes can cause big financial losses, therefore the need for identifying and correcting the errors has become critical. Business process analysis is often performed by walkthroughs only. Many business process tools have a built-in simulator, which is usually used for performance analysis (e.g. the Aris Toolset). Simulations can also be used for model validation and testing, but verification is needed to guarantee behavioral properties like compliance w.r.t. law regulations. Real-life business processes are too large and complex to be verified manually, and automated support is therefore essential.

There are many verification techniques developed for embedded systems and hardware, which share many features with business processes: they all are concurrent systems where resources play an essential role. Therefore, verification techniques developed for them can be used for the verification of business processes as well. There are however a number of properties specific for business processes. One important correctness criterion is *proper termination*, which means that the workflow is always able to finish its job in a proper way and no garbage is left in the workflow after the workflow has terminated. Another important correctness criterion is *non-redundancy* of the tasks of the workflow which means that all the tasks of the workflow are potentially used in the execution.

For the verification of business processes a special class of Petri nets has proven to be useful: the workflow nets [11]. Workflow nets have a special structure: there is one initial place (with no input transitions) and one final place (with no output transitions) and all other nodes are on a path from the initial place to the final place. Workflow nets have been extensively studied in the

literature [9, 59, 60, 133, 134]. One very important sanity check for workflow nets is *soundness* [3], which corresponds to a combination of proper termination and non-redundancy of transitions.

From the verification point of view, (high-level) Petri nets offer a variety of verification techniques. Structural properties, e.g. presence of siphons and traps, can be checked directly on the Petri net and have the advantage of low computational complexity. Behavioral properties, e.g. reachability, boundedness, deadlock-freedom and home markings require often the analysis of the whole behavior of the net (represented by the set of reachable markings), which can be infinite. In spite of the large expressive power of Petri nets, most behavioral properties of interest are decidable [51]. However they tend to have high complexity. For many extensions of Petri nets, however, most of the behavioral properties become undecidable.

In the remainder we discuss the main directions in tackling the verification problem.

Model checking

Exhaustive verification, also called model checking [33, 34, 115], has been successfully applied to the verification of finite state systems. The main advantage of model checking is that it can be fully automated and it can reproduce erroneous behavior (counterexamples) when the requirements are violated. Petri nets have the advantage that model checking is decidable for many interesting properties even when the state space is infinite [51]. For (high-level) Petri nets with finite state space, model checking properties expressed in temporal logics (e.g. LTL and CTL) has been thoroughly investigated [125, 87].

The main problem with the application of model checking in practice is the state space explosion. Several additional methods have been proposed to alleviate its effects.

Reduction

Reductions are transformations of nets that reduce the size of the state space or of the net while preserving the properties to be verified. Reductions are mainly used as an optimization technique before applying exhaustive verification. Ideally, reductions should produce a net that is smaller than the original net and has the "same" behavior, i.e. has the same behavioral properties. For instance, symmetry reduction [125] and fusion and replacement under place bisimulation [126] preserve branching-time behavior. Structural reduction techniques preserving such behavioral properties as boundedness and liveness [22, 21, 57, 22, 44] have been studied thoroughly for Petri nets .

Abstraction

Abstractions [39, 36] define a finite abstract state domain and a transition relation between the abstract states, which is an approximation of a given behavior

of a system. An abstraction function defines the relation between an abstract state and its corresponding concrete state(s). Depending on the abstraction function, the model can have less behavior than the original system (underapproximation) or more behavior than the original system (overapproximation). There are three types of property preservation: *weak preservation* (when every property that is true on the abstract model is true on the concrete model), *strong preservation* (when the same properties hold on both the abstract model and the concrete model) and *error preservation* (when a property that is violated on the abstract model is also violated on the concrete model). For Petri nets, certain abstract domains, e.g. convex polyhedra, polynomials [32], downward closed sets (for the Karp-Miller coverability tree [55]) have been successfully applied for proving several properties of nets, like deadlock-freedom and boundedness.

Compositional verification

Compositional verification has been proposed as a natural way of taking advantage of the modular design for large nets. Traditionally, there are two ways to tackle compositional verification: the bottom-up approach and the top-down approach. In the bottom-up approach, also called *modular abstraction*, a net is decomposed into subnets, and each component is model checked separately, eventually abstracting its environment (the subnets with which it interacts). Subsequently, following a divide-and-conquer reasoning, properties of the whole system can be derived from checking properties of the abstracted subnets. The complementary approach — *compositional refinement* — starts from some abstract/underspecified net, and by stepwise refinement replaces parts of the original net by more detailed net descriptions.

The multitude of verification techniques enumerated above place Petri nets as relevant candidates for approaching different issues in modeling and analysis in the business processes domain.

1.2 Research Questions

In this section we identify important issues in business process modeling and verification that we address in this thesis.

Modular verification of soundness

For large workflow nets, verification can be very expensive. For workflow nets, the parametrized notion of soundness, called *generalized soundness* was introduced to build workflows by using refinements [59, 60] in a compositional way. Generalized soundness guarantees that every marking reachable from an initial marking with k tokens on the initial place terminates properly, i.e. it can reach a marking with k tokens on the final place, for an arbitrary natural number

k. An expensive decision procedure for generalized soundness was described in [60].

The first research question is to integrate different verification techniques in order to verify proper termination (soundness) of very large workflows in an efficient way. In particular we are interested in developing a more efficient algorithm for the verification of generalized soundness.

Modeling and verification of adaptive workflow

Languages for modeling adaptive business processes have appeared due to the need of companies to have more flexibility in their business environment as opposed to classical workflow, where the structure of the business process is known at design time. A typical example of an adaptive workflow is the medical guideline-based workflow, where decisions and process changes made at the level of a doctor have to be transferred to the medical assistant level. Another important aspect of medical workflows is exception handling [58, 96]. For instance, an exceptional situation at a medical assistant level must be reported immediately to the doctor level.

Our objective is to propose a method for modeling and verification of adaptive workflows appropriate for applications in healthcare.

Modeling and verification of time and data-dependent industrial business processes

Extended event-driven process chains (eEPCs) form a language for modeling business processes in ARIS and SAP, documenting industrial reference models and designing workflows. Like many other business process languages in industry, the semantics of eEPCs are given by the simulator (ARIS ToolSet). Several formalizations of the EPC semantics were made [2, 86, 78], but none of them takes data and time into account.

Our research goal is to perform verification of eEPCs by taking data and time aspects into account. In particular, we are interested in a translation to colored Petri nets for performing verification of eEPCs with data and time.

Modeling history-dependent business processes

Nowadays, most workflow management systems record events in event logs, also known as transaction logs, or audit trails. Logs are naturally used to monitor running business processes for performance analysis and decision making, where decisions depend on the previous execution of the workflow [117, 8]. Our objective is to specify formally and evaluate history-based guards in business processes. As a side question, we would like to find the most common patterns for specifying such guards.

Figure 1 shows an overview of the reseach questions we are tackling in this thesis.

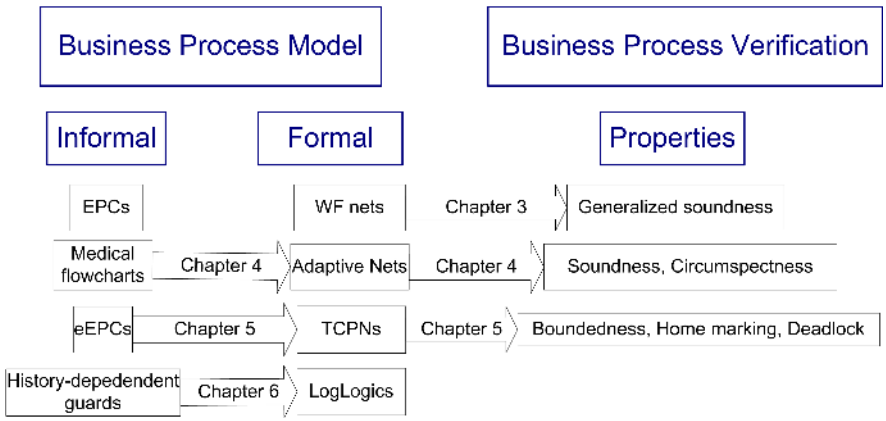


Fig. 1: Overview of the research topics

1.3 Related Work

In this section, we review some related work. Each separate chapter will discuss relevent literature in more detail.

Modular verification of soundness

Many classes of nets that preserve properties by refinement/abstraction were introduced, e.g. well-formed nets [135], well-behaved nets [129], modules [26], open interface nets [27] and re-entrant nets [28]. Most of these results focus on defining properties that subnets should possess in order to preserve general properties like boundedness, liveness or deadlock-freedom, and discuss the decidability status of those properties and not the practical application to specific properties of business processes. Other approaches, e.g. incremental verification [90] and modular verification [88], focus on finding better representations of modular state spaces with respect to some temporal or net property to be verified. For workflow nets, a parametrized notion of soundness (generalized soundness) was introduced to verify soundness of workflows by using refinement [59, 60] in a compositional way. Furthermore, in [60] a procedure is described that uses abstraction for the check of generalized soundness for workflow nets. Workflow soundness check is implemented in Woflan [138].

Modeling and verification of adaptive workflow

Adaptive workflow concepts have been introduced due to the need of companies to support dynamic changes in their business processes [50]. Many formalisms and corresponding compliance criteria have been proposed in order to cope with the so-called “dynamic change bug” that appears when transferring the state of the old workflow to the state of the changed workflow [4, 142, 121]. There

are several trends: total change of the structure of the workflow with history-based compliance [121] and structured change using Petri net transformations that preserve properties of the initial workflow [5]. Another approach to deal with adaptive workflows was proposed in [109], where the user can modify behavior at runtime by introducing new requirements specified in a graphical language based on the linear time logic (LTL). These approaches struggle with large workflows, where separation of concerns and reuse of existing workflows is crucial. A lot of attention has been devoted to adaptive medical workflow with exception handling [58, 96] in the attempt to implement adaptive workflow using the object-oriented approach. However, these approaches do not support any verification of workflows.

Modeling and verification of time and data-dependent business processes

Many industrial languages are very close or even inspired by Petri nets, e.g. UML activity diagrams [48] and EPCs [41]. There exist workflow management systems that use (extensions of) Petri nets as a process modeling language, e.g. ARIS, Cosa and Staffware. There are academic tools which use (colored) Petri nets within WFMS for simulation purposes, e.g. CPN Tools [37], Exspect [13] (used in Protos) and Yasper [63]. Since simulation cannot cover all the behavior of a business process, not all errors can be found automatically.

There are many translations of informal business process languages to formal languages for verification purposes, e.g. WS-BPEL to Petri nets [71] verified with LoLA [124], EPCs without data and time to Petri nets [2, 43] verified with Woflan [138] and LoLA [124]. However most of them concentrate on the control flow and "abstract" from data and time. Abstractions from data and time are not reliable since they result in both removal and addition of behavior. Therefore performing verification on the "abstraction" can give false errors which cannot be encountered in the model with data and time for some models, and positive verification report for other models whose original is erroneous.

Modeling history-dependent business processes

Most WFMSs typically support logging capabilities that register the execution of business processes by means of audit trails or history logs. There are several purposes of using logs: for monitoring and administration, for performance analysis, redesign of business processes and process mining [83], for the support of dynamic changes in adaptive workflows [121], and for control-flow decisions during workflow executions [105]. Most approaches for querying logs at runtime use database querying methods (e.g. [105, 83]), which is not suitable for the verification of temporal properties. Moreover, timing of logged events is essential, whereas in database languages there are very few constructs to deal with it. The suitability of using Petri nets with history logs was motivated in [66].

1.4 Overview

Chapter **2** provides a review of some mathematical notions and some notions from computer science that are used in the rest of the thesis.

In Chapter **3**, we improve the decision procedure from [60] for the problem of generalized soundness of workflow nets. We report on experimental results obtained with the prototype implementation we made. Furthermore, we show how this procedure can be used to prove soundness by reduction of generalized sound subworkflow nets. Since the generalized soundness check is still expensive, we show how we can use transformation techniques that preserve liveness and boundedness of the workflow and incorporate these transformations in the soundness check for large workflows.

In Chapter **4**, we consider adaptive workflow nets, a subclass of nested nets that allows more comfort and expressive power for modeling adaptation and exception handling in workflow nets. We illustrate our new modeling approach by choosing a running example from the healthcare domain, which is a typical domain for adaptive workflow. We define several behavioral properties of adaptive workflow nets, among which soundness and proper handling of exceptions. For adaptive workflows, we employ compositional verification of soundness and circumspectness by using abstractions.

In Chapter **5** we consider Extended Event-driven Process Chains (eEPCs), a language which is widely used for modeling business processes, documenting industrial reference models and designing workflows. We describe how to translate eEPCs into timed colored Petri nets in order to verify processes given by eEPCs with the CPN Tools.

Chapter **6** treats choices in business processes that are based on the process history saved as a log-file listing events and their time stamps. We introduce *LogLogics*, a finite-path variant of the Timed Propositional Temporal Logic with Past, which can be in particular used for specifying guards in business process models. We reduce the check of the truth value of a *LogLogics* formula to a check on a finite abstraction and present an evaluation algorithm. We also define *LogLogics* patterns for commonly occurring properties.

Chapter **7** draws general conclusions and gives directions for future work.

2

Preliminaries

This chapter reviews some mathematical notions and some notions from computer science that will be used in the remainder of the thesis.

2.1 Notions of Set and Graph Theory

We review here some basic notions of the set and graph theory (see [91] for more details).

Set theory

We use standard notation for sets and set operations. We recall some notations: the *empty set* is denoted by $\varnothing$, *element inclusion* by $\in$, *set intersection* by $\cap$, *set union* by $\cup$, *set difference* by $\setminus$, *set inclusion* by $\subseteq$ and *strict set inclusion* is denoted by $\subset$.

We denote the set of natural numbers by $\mathbb{N}$, the set of non-zero natural numbers by $\mathbb{N}^+ \stackrel{\text{def}}{=} \mathbb{N} \setminus \{0\}$, the set of integers by $\mathbb{Z}$, the set of rational numbers by $\mathbb{Q}$, the set of non-negative rational numbers by $\mathbb{Q}^+$, the set of real numbers by $\mathbb{R}$, and the set of positive real numbers by $\mathbb{R}^+$.

Definition 2.1. [Binary relation]
A binary relation R *over two sets* S *and* S' *is a subset of* $S \times S'$. *The* domain *of the relation* R, *denoted by* $Dom(R)$, *is given by* $Dom(R) \stackrel{\text{def}}{=} \{a \in S | \exists b \in S' : (a,b) \in R\}$ *and the* codomain *of* R, *denoted by* $Ran(R)$, *is given by* $Ran(R) \stackrel{\text{def}}{=} \{b \in S' | \exists a \in S : (a,b) \in R\}$. *If* $S = S'$, *we say that* R *is a* relation over S. *We write* $a\ R\ b$ *iff* $(a,b) \in R$.

Let R, R' *be binary relations over* S.

- $R^0 = \{(s,s) | s \in S\}$ *is called the* identity relation *over* S;
- R^{-1} *denotes the* converse relation *of* R, *i.e.* $R^{-1} \stackrel{\text{def}}{=} \{(b,a) | (a,b) \in R\}$;
- $R \cup R' \stackrel{\text{def}}{=} \{(a,b) | (a,b) \in R \vee (a,b) \in R'\}$ *is the* union *of* R *and* R';
- $R \circ R' \stackrel{\text{def}}{=} \{(a,c) | \exists b \in S : (a,b) \in R \wedge (b,c) \in R'\}$ *is the* composition *of the relations* R *and* R'.
- $R^{j+1} \stackrel{\text{def}}{=} R^j \circ R$, *for* $j \geq 0$;
- $R^+ \stackrel{\text{def}}{=} \bigcup_{j \geq 1} R^j$ *is the* transitive closure *of* R;
- $R^* \stackrel{\text{def}}{=} \bigcup_{j \geq 0} R^j$ *is the* transitive reflexive closure *of* R;
- $(R \cup R^{-1})^*$ *is the* symmetric, reflexive and transitive closure *of* R.

A binary relation f over A and B is called a *partial function* if

$$(\forall a_1, a_2 \in A \forall b_1, b_2 \in B)((a_1, b_1) \in f \wedge (a_2, b_2) \in f \wedge a_1 = a_2 \Rightarrow b_1 = b_2).$$

We write $f : A \nrightarrow B$ iff f is a partial function with $Dom(f) \subseteq A$ and $Ran(f) \subseteq B$ and $f(a) = b$ iff $(a,b) \in f$. A partial function $f : A \nrightarrow B$ is called a *function*, denoted by $f : A \to B$ if $Dom(f) = A$.

Let $f : A \to B$ be a function and $A' \subseteq A$. We define *the restriction of* f *to* A' as the function $f_{|A'} : A' \to B$ such that for all $a \in A'$, $f'(a) = f(a)$.

Let $f\colon A \nrightarrow B$ be a partial function, $a \in A$ and $b \in B$. We define the (partial) function $f[a \mapsto b]\colon A \nrightarrow B$ as:

$$f[a \mapsto b](y) = \begin{cases} b & \text{if } y = a; \\ f(y) & \text{if } y \in Dom(f) \setminus \{a\}. \end{cases}$$

Definition 2.2. [Bag]
A bag (multiset) *over a non-empty finite set S is a mapping $m\colon S \to \mathbb{N}$. The non-negative number $m(s) \in \mathbb{N}$ denotes the number of occurrences of the element $s \in S$ in bag m. The set of all bags over S is $\mathbb{N}^S$.*

Let $m, m' \in \mathbb{N}^S$.

- $m + m'\colon S \to \mathbb{N}$ is the sum of bags m and m' defined as $(m + m')(s) \stackrel{\text{def}}{=} m(s) + m'(s)$ for all $s \in S$;
- $m - m'\colon S \to \mathbb{N}$ is the difference of bags m and m' defined as $(m - m')(s) \stackrel{\text{def}}{=} \max\{0, m(s) - m'(s)\}$ for all $s \in S$;
- $m = m'$ iff $m(s) = m'(s)$ for all $s \in S$;
- $m \leq m'$ iff $m(s) \leq m'(s)$ for all $s \in S$;
- $m < m'$ iff $m \leq m'$ and $m \neq m'$.

We overload $\varnothing$ and $\in$ to denote the *empty bag* and *element inclusion*, i.e. $m = \varnothing$ if $m(s) = 0$ for any $s \in S$ and $s \in m$ iff $m(s) > 0$, respectively. We write $m = 2[a] + [b]$ for a bag m with $m(a) = 2$, $m(b) = 1$ and $m(x) = 0$ for any $x \in S \setminus \{a, b\}$.

Definition 2.3. [sequence] *Let Σ be a finite set and $\epsilon \notin \Sigma$.*

- *A* finite sequence *of length $n \in \mathbb{N}^+$ over Σ is a mapping $\sigma\colon \{1, \ldots, n\} \to \Sigma$. We write $\tilde{a}$ for the sequence $\{1\} \to \{a\}$, for some $a \in \Sigma$.*
- *The* empty sequence *(of length 0) over Σ is the mapping $\sigma\colon \varnothing \to \Sigma$, denoted by ϵ.*
- *An* infinite sequence *over Σ is a mapping $\sigma\colon \mathbb{N}^+ \to \Sigma$ and we say that the length of an infinite sequence is ω (infinite), where we define $\omega > n$, for all $n \in \mathbb{N}$.*

We denote the length of a sequence σ by $|\sigma|$.

We denote the set of all finite sequences (of length $n \geq 0$) over Σ by Σ^*.

The *concatenation* of two sequences $\sigma, \sigma' \in \Sigma^*$, denoted by $\sigma\sigma'$, is ϵ if $\sigma = \sigma' = \epsilon$ and the sequence $\sigma\sigma'\colon \{1, \ldots, |\sigma| + |\sigma'|\} \to \Sigma$ otherwise, where

$$(\sigma\sigma')(i) = \begin{cases} \sigma(i) & \text{if } 1 \leq i \leq |\sigma| \\ \sigma'(i - |\sigma|) & \text{if } |\sigma| < i \leq |\sigma| + |\sigma'|. \end{cases}$$

Let Σ and Σ' be two finite sets (alphabets). Let $h{:}\,\Sigma \to \Sigma'^*$ be a function assigning to each element of $a \in \Sigma$ a finite word σ over Σ'. The *homomorphic extension* of h to Σ'^*, is the function $h{:}\,\Sigma^* \to \Sigma'^*$ defined by $h(\epsilon) = \epsilon$ and $h(\sigma\sigma') = h(\sigma)h(\sigma')$ for all $\sigma, \sigma' \in \Sigma^*$.

We define the *projection* from sequences over Σ to sequences over $\Sigma_1 \subseteq \Sigma$ as a homomorphic extension of $\sigma_{|\Sigma_1}{:}\,\Sigma \to \Sigma_1^*$ such that $h(\tilde{a}) = \epsilon$ for all $\tilde{a} \in \Sigma \setminus \Sigma_1$ and $h(\tilde{a}) = \tilde{a}$ for all $a \in \Sigma_1$.

Lemma 2.1. [Dickson [46]]
Every infinite set of bags over a finite set contains an infinite increasing (w.r.t. $<$) sequence.

Graph Theory

Definition 2.4. [directed graph]
A directed graph *is a tuple $G = (V, A)$, where V is a set of* vertices *and $A \subseteq V \times V$ is a set of* arcs. *Every arc $a \in A$ is a pair $(v_1, v_2) \in V \times V$ consisting of the* input *vertex v_1 and the* output *vertex v_2.*

Definition 2.5. [bipartite graph]
A bipartite graph *is a directed graph $G = (V_1 \cup V_2, A)$, where $V_1 \cap V_2 = \emptyset$ and $A \subseteq (V_1 \times V_2) \cup (V_2 \times V_1)$.*

Definition 2.6. [multigraph]
A multigraph *is a tuple $G = \langle V, A, W \rangle$, where (V, A) is a graph and W is a bag of arcs over A.*

Definition 2.7. [path]
A path σ of length $n \in \mathbb{N}$ *in a graph $G = (V, A)$ is a finite sequence of vertices $\sigma = v_1 \dots v_n$ ($v_j \in V$ for $j = 1, \dots, n$) such that each pair $(v_j, v_{j+1}) \in A$ ($j = 1, \dots, n-1$). We call n the* length *of path σ and denote it by $|\sigma|$. The* empty path *is the empty sequence ϵ that has the length $|\epsilon| = 0$.*

We denote the set of all finite, possibly empty, paths by V^* and the set of all finite non-empty paths by V^+.

A path σ is a *prefix* of a path γ if there is a path σ' such that $\gamma = \sigma\sigma'$.

Definition 2.8. [weakly connected, strongly connected]
A graph $G = (V, A)$ is weakly connected *if for every two vertices $v_1, v_2 \in V$, $(v_1, v_2) \in (A \cup A^{-1})^*$. A graph $G = (V, A)$ is* strongly connected *if for every two vertices $v_1, v_2 \in V$, $(v_1, v_2) \in A^*$.*

We denote the set of *output vertices* of $v \in V$ as $v^\bullet$, i.e. $v^\bullet \stackrel{\text{def}}{=} \{v' | (v, v') \in A\}$. Similarly, ${}^\bullet v \stackrel{\text{def}}{=} \{v' | (v', v) \in A\}$ is the set of *input vertices* of $v \in V$. Given a set of vertices $X \subseteq V$, we define ${}^\bullet X \stackrel{\text{def}}{=} \bigcup_{v \in X} {}^\bullet v$ and $X^\bullet \stackrel{\text{def}}{=} \bigcup_{v \in X} v^\bullet$. We also write $v^\bullet = y$, when $v^\bullet = \{y\}$.

We denote the set of *ingoing arcs* of a vertex $v \in V$ as A_v^{in}, i.e. $A_v^{in} = \{(x,v)|x \in {}^\bullet v\}$ and the set of *outgoing arcs* of v as A_v^{out}, i.e. $A_v^{out} = \{(v,x)|x \in v^\bullet\}$. In case A_v^{in} is singleton, a_v^{in} denotes the ingoing arc of the vertex v. Similarly, if A_v^{out} is singleton, a_v^{out} denotes the outgoing arc of the vertex v.

2.2 Transition Systems and Behavioral Equivalences

In this section, we define labeled transition systems and semantic equivalences on them. First we provide some necessary definitions about transition systems and their traces and then introduce some equivalence relations.

Definition 2.9. [Labeled transition system]
A labeled transition system *is a tuple* $U = \langle S, \Sigma, \rightarrow, s_0 \rangle$ *where* S *is a set of* states, Σ *is a finite set of* labels, $\rightarrow \subseteq S \times \Sigma \times S$ *is a* transition relation *and* $s_0 \in S$ *is an initial state.*

We denote $(s_1, a, s_2) \in \rightarrow$ as $s_1 \xrightarrow{a} s_2$, and we say that s_2 is reachable from s_1 by an action labeled by a. For a sequence of labels $\sigma = a_1 \cdots a_n \in \Sigma^*$ we write $s_1 \xrightarrow{\sigma} s_2$ when $s_1 = s^0 \xrightarrow{a_1} s^1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s^n = s_2$, and $s_1 \xrightarrow{\sigma}$ when $s_1 \xrightarrow{\sigma} s_2$ for some s_2. In this case we say that σ is a trace of U. Finally, $s_1 \xrightarrow{*} s_2$ means that s_2 is reachable from s_1, i.e. there exists a sequence $\sigma \in \Sigma^*$ such that $s_1 \xrightarrow{\sigma} s_2$.

When analyzing a transition system, we may vary the level of detail at which it is observed. We thus distinguish between visible and invisible behavior. We use the label $\tau \in \Sigma$ to denote invisible actions and write $s_1 \Rightarrow s_2$ when $s_1 = s_2$ or $s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_2$. We write $s_1 \xRightarrow{a} s_2$ if $s_1 \Rightarrow s_1' \xrightarrow{a} s_2' \Rightarrow s_2$. For a sequence of labels $\sigma = a_1 \cdots a_n \in \Sigma^*$ we write $s_1 \xRightarrow{\sigma} s_2$ when $s_1 = s^0 \xRightarrow{a_1} s^1 \xRightarrow{a_2} \ldots \xRightarrow{a_n} s^n = s_2$, and $s_1 \xRightarrow{\sigma}$ when $s_1 \xRightarrow{\sigma} s_2$ for some s_2. In this case we say that σ is a trace of U. Finally, $s_1 \xRightarrow{*} s_2$ means that s_2 is reachable from s_1, i.e. there exists a sequence $\sigma \in \Sigma^*$ such that $s_1 \xRightarrow{\sigma} s_2$.

To compare the behavior of two transition systems a number of equivalences have been proposed in the literature (see [56] for an overview).

Definition 2.10. [strong trace equivalence]
The strong trace set *of a labeled transition system* U *is* $\{\sigma \in \Sigma^* | s_0 \xrightarrow{\sigma}\}$. *Two labeled transition systems are* strongly trace equivalent *iff their strong trace sets are equal.*

Definition 2.11. [weak trace equivalence]
The weak trace set *of a labeled transition system* U *is* $\{\sigma \in (\Sigma \setminus \{\tau\})^* | s_0 \xRightarrow{\sigma}\}$. *Two labeled transition systems are* weakly trace equivalent *iff their weak trace sets are equal.*

Definition 2.12. [SIMULATION]
Let $U_1 = \langle S_1, \Sigma, \rightarrow_1, s_0^1 \rangle$ *and* $U_2 = \langle S_2, \Sigma, \rightarrow_2, s_0^2 \rangle$ *be two labeled transition systems. A relation* $R \subseteq S_1 \times S_2$ *is a* simulation *iff for all* $s_1, s_1' \in S_1$, $s_2 \in S_2$ *such that* $s_1 \; R \; s_2$ *and* $s_1 \stackrel{a}{\rightarrow}_1 s_1'$, *there exists* $s_2' \in S_2$ *such that* $s_2 \stackrel{a}{\rightarrow}_2 s_2'$ *and* $s_1' \; R \; s_2'$, *and moreover* $s_0^1 \; R \; s_0^2$. *We say that* U_2 *can simulate* U_1 *if there exists such a simulation relation* R *between them.*

Definition 2.13. [WEAK SIMULATION]
Let $U_1 = \langle S_1, \Sigma, \rightarrow_1, s_0^1 \rangle$ *and* $U_2 = \langle S_2, \Sigma, \rightarrow_2, s_0^2 \rangle$ *be two labeled transition systems. A relation* $R \subseteq S_1 \times S_2$ *is a* weak simulation *iff for all* $s_1, s_1' \in S_1$, $s_2 \in S_2$ *such that* $s_1 \; R \; s_2$ *and* $s_1 \stackrel{a}{\Rightarrow}_1 s_1'$, *there exists* $s_2' \in S_2$ *such that* $s_2 \stackrel{a}{\Rightarrow}_2 s_2'$ *and* $s_1' \; R \; s_2'$, *and moreover* $s_0^1 \; R \; s_0^2$. *We say that* U_2 *weakly simulates* U_1 *if there exists such weak simulation relation* R *between them.*

Definition 2.14. [BRANCHING SIMULATION]
Let $U_1 = \langle S_1, \Sigma, \rightarrow_1, s_0^1 \rangle$ *and* $U_2 = \langle S_2, \Sigma, \rightarrow_2, s_0^2 \rangle$ *be two labeled transition systems. A relation* $R \subseteq S_1 \times S_2$ *is a* branching simulation *iff* $s_0^1 \; R \; s_0^2$ *and for all* $s_1, s_1' \in S_1$, $s_2 \in S_2$ *such that* $s_1 \; R \; s_2$ *and* $s_1 \stackrel{a}{\rightarrow}_1 s_1'$, *either*

1. $a = \tau$ *and* $s_1' \; R \; s_2$ *or*
2. *there exist* $s^1, \ldots, s^n, s_2' \in S_2$ *such that* $s_2 \stackrel{\tau}{\rightarrow}_2 s^1 \stackrel{\tau}{\rightarrow}_2 \ldots \stackrel{\tau}{\rightarrow}_2 s^n \stackrel{a}{\rightarrow}_2 s_2'$, $s_1 \; R \; s^i$ *for all* $i = 1..n$, *and* $s_1' \; R \; s_2'$.

Definition 2.15. [BISIMULATION]
Let $U_1 = \langle S_1, \Sigma, \rightarrow_1, s_0^1 \rangle$ *and* $U_2 = \langle S_2, \Sigma, \rightarrow_2, s_0^2 \rangle$ *be two labeled transition systems. A relation* $R \subseteq S_1 \times S_2$ *is a* bisimulation *iff* R *and* R^{-1} *are simulations. We say that* U_1 *and* U_2 *are* bisimilar *and write* $U_1 \leftrightarroweq_s U_2$ *iff there exists a bisimilation* R *between them.*

Definition 2.16. [WEAK BISIMULATION]
Let $U_1 = \langle S_1, \Sigma, \rightarrow_1, s_0^1 \rangle$ *and* $U_2 = \langle S_2, \Sigma, \rightarrow_2, s_0^2 \rangle$ *be two labeled transition systems. A relation* $R \subseteq S_1 \times S_2$ *is a* weak bisimulation *if* R *and* R^{-1} *are weak simulations. We say that* U_1 *and* U_2 *are* weakly bisimilar *and write* $U_1 \leftrightarroweq_w U_2$ *iff there exists a weak bisimilation* R *between them.*

Definition 2.17. [BRANCHING BISIMULATION]
Let $U_1 = \langle S_1, \Sigma, \rightarrow_1, s_0^1 \rangle$ *and* $U_2 = \langle S_2, \Sigma, \rightarrow_2, s_0^2 \rangle$ *be two labeled transition systems. A relation* $R \subseteq S_1 \times S_2$ *is a* branching bisimulation *iff if* R *and* R^{-1} *are branching bisimulations. We say that* U_1 *and* U_2 *are* branching bisimilar *and write* $U_1 \leftrightarroweq_{br} U_2$ *iff there exists a branching bisimilation* R *between them.*

2.3 Petri Nets

This section presents an introduction to Petri nets as a formalism for modeling and analysis of concurrent systems. Further, some basic structural and behavioral properties of nets that are used in this thesis are defined (see [110, 104, 118] for more details).

Petri nets were introduced by Carl Adam Petri in [111] and have been intensively applied in many fields like communication, manufacturing, hardware and software engineering, and business process management (see [45] for a review).

Definition 2.18. [(Labeled) Petri nets]
A Petri net *is a tuple* $N = \langle P, T, F^+, F^- \rangle$*, where*

- P *and* T *are two disjoint non-empty finite sets of* places *and* transitions *respectively;*
- F^+ *and* F^- *are mappings* $(P \times T) \to \mathbb{N}$ *called* flow functions *from transitions to places and from places to transitions respectively.*

A labeled Petri net *is a tuple* $\langle P, T, F^+, F^-, l \rangle$*, where* $\langle P, T, F^+, F^- \rangle$ *is a Petri net and* $l : T \to \Sigma$ *is a labeling function mapping each* $t \in T$ *to some label* $l(t) \in \Sigma$*, where* Σ *is a finite label set.*

A Petri net can be represented graphically as a bipartite multigraph, i.e. a directed multigraph where the set of vertices is partitioned into two sets (places and transitions). When the multiplicity of the arcs is 1, i.e. F^+ and F^- are defined on $(P \times T) \to \{0, 1\}$, we also write $N = \langle P, T, A, l \rangle$, where $A \subseteq (P \times T) \cup (T \times P)$ is the set of arcs between places and transitions and conversely, called the *flow relation* of N.

Definition 2.19. [incidence matrix]
Let $N = \langle P, T, F^+, F^- \rangle$ *be a Petri net. We say that* $F = F^+ - F^-$ *is the* incidence matrix *of net* N*.*

Definition 2.20. [Parikh vector]
Let $N = \langle P, T, F^+, F^- \rangle$ *be a Petri net and* σ *be a sequence of transitions. The* Parikh vector $\overrightarrow{\sigma} : T \to \mathbb{N}$ *of* σ *maps every transition* t *of* σ *to the number of occurrences of* t *in* σ*.*

We use the following notations:

- $(p^\bullet)_N \stackrel{\text{def}}{=} \sum_{t \in T} F^-(p, t)[t]$ is the bag of *output transitions* of a place $p \in P$;
- $(t^\bullet)_N \stackrel{\text{def}}{=} \sum_{p \in P} F^+(p, t)[p]$ is the bag of *output places* of a transition $t \in T$;
- $(Q^\bullet)_N \stackrel{\text{def}}{=} \bigcup_{p \in Q} \{t | t \in p^\bullet\}$ is the set of *output transitions* of a set of places $Q \subseteq P$ in N;
- $({}^\bullet p)_N \stackrel{\text{def}}{=} \sum_{t \in T} F^+(p, t)[t]$ denotes the bag of *input transitions* of a place $p \in P$ in N;

- $({}^\bullet t)_N \stackrel{\text{def}}{=} \sum_{p \in P} F^-(p,t)[t]$ denotes the bag of *input places* of a transition $t \in T$ in N;
- $({}^\bullet Q)_N \stackrel{\text{def}}{=} \bigcup_{p \in Q} \{t | t \in {}^\bullet p\}$ the set of input transitions of a set of places $Q \subseteq P$ in N.

We write ${}^\bullet p$ (${}^\bullet t$), $p^\bullet$ ($t^\bullet$), ${}^\bullet Q$ and $Q^\bullet$, respectively when N is clear from the context. A place p with ${}^\bullet p = \varnothing$ is called a *source place* and a place q with $q^\bullet = \varnothing$ is called a *sink place*.

The state (configuration) of a net is given by its *marking* that represents the distribution of tokens in the places:

Definition 2.21. [MARKING, MARKED NET]
A marking *of a (labeled) Petri net N is a mapping $m\colon P \to \mathbb{N}$ that assigns a number of tokens to each place $p \in P$. A (labeled)* marked Petri net *is a tuple (N, m), where N is a (labeled) Petri net and m is a marking.*

We interpret markings both as P-dimensional column vectors and as bags over P. Let $p \in P$; then $\bar{p}$ denotes the vector such that $\bar{p}(p) = 1$ and $\bar{p}(p') = 0$ for all $p' \in P$ such that $p' \neq p$. Let the vector $\bar{0}$ represent the *zero vector (marking)* whose length is context-defined, i.e. $\bar{0}(i) = 0$, for all $i \in \{1, \ldots, |P|\}$. Similarly, we interpret the flow functions as bags over $P \times T$ and as $|P| \times |T|$ matrices over $\mathbb{N}$ with the usual matrix operations.

Definition 2.22. [ENABLING CONDITION, FIRING]

1. *A transition $t \in T$ is* enabled *in a marking m, denoted by $m \xrightarrow{t}$, if $F^-(p,t) \leq m(p)$, for all $p \in P$.*
2. *If t is enabled in a marking m, t may* fire *yielding a new marking m', denoted by $m \xrightarrow{t} m'$, where $m'(p) = m(p) - F^-(p,t) + F^+(p,t)$, for all $p \in P$.*

We extend the firing rule homomorphically to sequences of transitions $\sigma \in T^*$, denoted by $m \xrightarrow{\sigma} m'$:

- $m \xrightarrow{\epsilon} m$ for all m;
- if $m \xrightarrow{\sigma} m'$ and $m' \xrightarrow{t} m''$ for some markings m, m', m'', $\sigma \in T^*$ and $t \in T$, then $m \xrightarrow{\sigma t} m''$.

Let (N, m) be a marked labeled Petri net. We extend the labeling function l homomorphically to sequences of transitions, i.e. $l(\sigma) = l(t_1) \ldots l(t_n)$ for some $\sigma = t_1 \ldots t_n \in T^*$ and $n \geq 1$.

Definition 2.23. [REACHABLE MARKING]
We say that a marking m' is reachable *from a marking m and write $m \xrightarrow{*} m'$ if there exists $\sigma \in T^*$ such that $m \xrightarrow{\sigma} m'$.*

We denote the set of all markings of N reachable from m by $\mathcal{R}(N, m)$, i.e. $\mathcal{R}(N, m) \stackrel{\text{def}}{=} \{m' | m \xrightarrow{*} m'\}$. Similarly, $\mathcal{S}(N, m)$ denotes the set of markings

of N that can reach m, i.e. $\mathcal{S}(N,m) \stackrel{\text{def}}{=} \{m' | m' \stackrel{*}{\longrightarrow} m\}$. The set of markings reachable from (that can reach) m by firings of transitions from $T' \subseteq T$ is denoted by $\mathcal{R}_{|T'}(N,m)$ ($\mathcal{S}_{|T'}(N,m)$).

The transition system of (N,m) is $\langle \mathcal{R}(N,m), T, \longrightarrow, m \rangle$, which is also known as the reachability graph of (N,m).

Definition 2.24. [HOME MARKING, HOME SPACE]
Let (N,m_0) be a marked Petri net, m a marking and M a set of markings of N. We say that m is a home marking *iff m is reachable from all markings of $\mathcal{R}(N,m_0)$. We say that M is a* home space *iff for every marking $m \in \mathcal{R}(N,m_0)$, there is a marking $m' \in M$ such that $m \stackrel{*}{\longrightarrow} m'$.*

Definition 2.25. [t-QUASI-LIVENESS, QUASI-LIVENESS]
A marked net (N,m_0) is t-quasi-live*, i.e. t is not* dead *iff there exists a marking $m \in \mathcal{R}(N,m_0)$ such that $m \stackrel{t}{\longrightarrow}$. A marked net (N,m_0) is* quasi-live *iff N is* t-quasi-live *for all $t \in T$.*

Definition 2.26. [t-LIVENESS, LIVENESS]
A marked net (N,m_0) is t-live *iff for all markings $m \in \mathcal{R}(N,m_0)$ there exists a marking m' such that $m \stackrel{*}{\longrightarrow} m'$ and $m \stackrel{t}{\longrightarrow}$. A marked net (N,m_0) is* live *iff (N,m_0) is t-live for all $t \in T$.*

Definition 2.27. [BOUNDEDNESS]
A marked net (N,m_0) is bounded *iff there exists $n \in \mathbb{N}$ such that for all $m \in \mathcal{R}(N,m_0)$, $m(p) \leq n$ for all $p \in P$.*

Proposition 2.1. [MARKING EQUATION] *[103]*
Let $N = \langle P, T, F^+, F^- \rangle$ be a Petri net. For every finite sequence σ of transitions such that $m \stackrel{\sigma}{\longrightarrow} m'$, the following equation holds: $m' = m + F \cdot \vec{\sigma}$.

Note that the reverse is not true: not every marking m' that can be represented as $m + F \cdot x$, for some $x \in \mathbb{N}^T$, is reachable from the marking m.

Definition 2.28. [TRAP, SIPHON] *[35]*
Let $N = \langle P, T, F^+, F^- \rangle$ be a Petri net. A subset of places $Q \subseteq P$ is called a trap *if $Q^\bullet \subseteq {}^\bullet Q$. A subset $Q \subseteq P$ is called a* siphon *if ${}^\bullet Q \subseteq Q^\bullet$. A trap or a siphon is called* proper *iff it is nonempty.*

Traps have the property that once marked they remain marked, whereas unmarked siphons remain unmarked whatever transition sequence occurs:

Proposition 2.2. [FUNDAMENTAL PROPERTIES OF TRAPS/SIPHONS] *[44]*
Let N be a Petri net and m be a marking of N.

Algorithm 1: *MaxSiphon*(N, P')

Input: N, $P' \subseteq P$
Output: $Q \subseteq P'$

$Q := P'$;
while *there exists a* $q \in Q$ *and* $t \in {}^\bullet q$ *such that* $t \notin Q^\bullet$ **do**
  $Q := Q \setminus \{q\}$
end
return Q

Algorithm 2: *MaxTrap*(N, P')

Input: N, $P' \subseteq P$
Output: $Q' \subseteq P'$

$Q' := P'$;
while *there exists a* $q \in Q'$ *and* $t \in q^\bullet$ *such that* $t \notin {}^\bullet Q$ **do**
  $Q' := Q' \setminus \{q\}$
end
return Q'

1. *Marked traps remain marked, i.e. if* Q *is a proper trap of* N *and* $\sum_{q \in Q} m(q) > 0$ *then* $\sum_{q \in Q} m'(q) > 0$ *for all* $m' \in \mathcal{R}(N, m)$.
2. *Unmarked siphons remain unmarked, i.e. if* Q *is a proper siphon of* N *and* $\sum_{q \in Q} m(q) = 0$ *then* $\sum_{q \in Q} m'(q) = 0$ *for all* $m' \in \mathcal{R}(N, m)$.

The existence of a siphon/trap within a given set of places can be checked using Starke's algorithm [128] which is polynomial in the size of the net. Algorithm 1 computes the largest siphon Q that is contained in a given set of places $P' \subseteq P$. The largest trap Q' contained in a given set of places $P' \subseteq P$ can be computed similarly (see Algorithm 2).

Definition 2.29. [Place invariants]*[89]*
Let $N = \langle P, T, F^+, F^- \rangle$ *be a Petri net. A place invariant is a row vector* $I : P \to \mathbb{Q}$ *such that* $I \cdot F = \bar{0}$. *The solutions to the equation* $I \cdot F = \bar{0}$ *form the space of all place invariants. We denote by* $\mathcal{I}$ *a matrix whose rows are vectors forming a* basis[1] *of the space of place invariants.*

We say that markings m and m' *agree on a place invariant* I if $I \cdot m = I \cdot m'$.

Proposition 2.3. [marking conservation]*[44]*
Let N *be a net and* m, m' *two markings such that* $m \xrightarrow{*} m'$. *Then* m_1 *and* m_2 *agree on all place invariants, i.e.* $\mathcal{I} \cdot m = \mathcal{I} \cdot m'$.

[1] A maximal linearly independent set of vectors

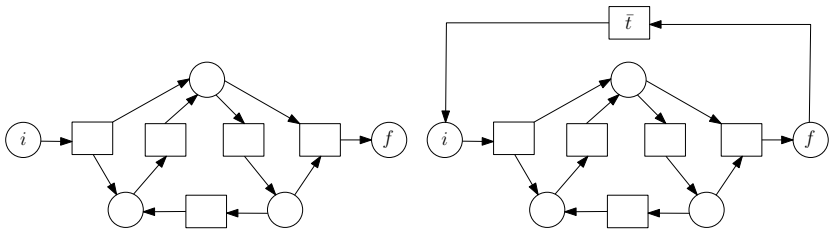


Fig. 2: Example of a sound WF net (left) and its closure (right)

2.4 Workflow Nets

Workflow management systems [9, 11] can be modeled by *workflow nets* (WF-nets), i.e. Petri nets with one initial and one final place and every place or transition being on a directed path from the initial to the final place.

Definition 2.30. [Workflow net]
A net N is a workflow net *(WF-net) iff*

- *N has two special places i and f. The* initial place *i is a source place, i.e. ${}^\bullet i = \varnothing$, and the* final place *$f$ is a sink place, i.e. $f^\bullet = \varnothing$.*
- *Every vertex $v \in P \cup T$ is on a path from i to f.*

The execution of a case is represented as a firing sequence that starts from the initial marking consisting of a single token on the initial place. The token on the final place with no garbage (tokens) left on the other places indicates the *proper termination* of the case execution. A workflow is called *sound* iff every reachable marking can terminate properly and all transitions can eventually fire.

Definition 2.31. [Soundness]*[9]*
A WF-net N is sound *iff*

1. *$\mathcal{R}(N, \bar{i}) \subseteq \mathcal{S}(N, \bar{f})$;*
2. *$(N, \bar{i})$ is quasi-live.*

Definition 2.32. [closure of a WF-net]*[9]*
The closure of a WF-net *$N = \langle P, T, F^+, F^- \rangle$ is a net $\bar{N} = \langle P, T \cup \{\bar{t}\}, \bar{F}^+, \bar{F}^- \rangle$, where $\bar{F}^-(i, \bar{t}) = \bar{F}^+(f, \bar{t}) = 0$, $\bar{F}^+(i, \bar{t}) = \bar{F}^-(f, \bar{t}) = 1$, $\bar{F}^+(p, t) = F^+(p, t)$ and $\bar{F}^-(p, t) = F^-(p, t)$ for all $(p, t) \in P \times T$.*

Figure 2 shows a sound WF net and its closure.

Lemma 2.2. *[9] A WF-net N is sound iff its closure $\bar{N}$ is live and bounded.*

2.5 Timed Colored Petri Nets

This section introduces the notion of timed colored Petri net [74].

Let $\mathcal{U}$ be a finite set of colors (types) which we call untimed and $\mathbb{T} \notin \mathcal{U}$ be a non-negative integer domain which we call time domain. A timed color (type) is a color $Y \times \mathbb{T}$, where $Y \in \mathcal{U}$. We denote the non-timed color of a timed type Y by $\pi_1(Y)$. Let *Const* be a finite set of constants and $\mathcal{V}$ a finite set of variables. Let *Expr* be a set of expressions built over the constants and variables from *Const* and $\mathcal{V}$ having the form $te = e@ + e'$, where e is an expression over variables and constants and $e' \in \mathbb{T}$. We denote the type of an expression $e \in Expr$ by $Type(e)$ and the set of variables of e by $Var(e)$. In the sequel, we denote the timed part of timed expressions te by te^t. If $te = e@ + 0$, we write $te = e$.

Definition 2.33. [TIMED BAG]
A timed bag *tm over a non-empty set S is a function $tm\colon (S \times \mathbb{T}) \to \mathbb{N}$ such that the sum $tm(s) \stackrel{\text{def}}{=} \sum_{\iota \in \mathbb{T}} tm(s,\iota)$ is finite for all $s \in S$.*

We denote the number of occurrences of the element $s \in S$ in the bag tm by $tm(s)$ and the time list of s by $tm[s] = [\iota_1, \iota_2, \ldots, \iota_{tm(s)}]$, i.e. $tm[s]$ contains the time values $\iota \in \mathbb{T}$ for which $tm(s,\iota) \neq 0$. Each ι appears $tm(s,\iota)$ times in the list, which is sorted such that $\iota_i \leq \iota_{i+1}$ for $1 \leq i \leq tm(s) - 1$. A timed bag is represented by $\sum_{s \in S} tm(s)'s@tm[s]$ and an untimed bag is represented as $\sum_{s \in S} tm(s)'s$. For an ordinary bag m and a time value ι, $m[s]^\iota$ denotes the timed bag obtained from $tm[s]$ by adding time stamp ι, i.e. $m[s]^\iota \stackrel{\text{def}}{=} \sum_{s \in S} m(s)'s@[\iota, \ldots, \iota]$. Similarly, $tm[s]^{+\iota}$ is the timed list obtained from $tm[s]$ by adding ι to all time stamps in the list and tm^{+r} is the bag obtained from tm where all timed elements of tm have their time stamps increased by ι.

We define comparison and subtraction of time lists l and l'. We write $l = [l_1, \ldots l_m] \leq l' = [l'_1, \ldots, l'_n]$ iff $m \leq n$ and $l_i \geq l'_i$ for all $i \in \{1, \ldots, m\}$. When $l \leq l'$, $l' - l$ is defined as a list of length $n - m$ obtained from l' in the following way: we remove from l' the largest element which is smaller than l_1; from the remaining list, we remove the largest time value which is smaller than l'_2, etc. For all timed bags tm_1, tm_2:

- $tm_1 \leq tm_2$ iff $\forall s \in S$, $tm_1[s] \leq tm_2[s]$;
- if $tm_1 \leq tm_2$, $tm_1 - tm_2 = \sum_{s \in S} (tm_1(s) - tm_2(s))'s@(tm_1[s] - tm_2[s])$.

Definition 2.34. [TCPN]
A timed colored Petri net (TCPN) is a tuple $\mathcal{N} = \langle P, T, A, \mathcal{E}, Gd, \mathcal{C}, cd, \mathcal{V}, l \rangle$, where

- *$\langle P, T, A, l \rangle$ is a labeled Petri net;*
- *$\mathcal{C}$ is a set of timed colors (types);*
- *$cd\colon P \to \mathcal{C}$ is a color mapping for places;*

- $\mathcal{E}$ *is an arc expression function which maps each arc to a bag of expressions which the same type as the color of the place of the arc, i.e. for all* $(p,t) \in (P \times T) \cap A$, $\mathcal{E}(p,t) \in \mathbb{N}^{\{e \in Expr | Type(e) = cd(p) \wedge e^t = 0\}}$ *and for all* $(t,p) \in (T \times P) \cap A$, $\mathcal{E}(t,p) \in \mathbb{N}^{\{e \in Expr | Type(e) = cd(p)\}}$;
- $Gd{:}T \to Expr$ *is a* guard function *satisfying* $Type(Gd(t)) = \{true, false\}$*; by default guards are set to true;*
- $\mathcal{V}$ *is a finite set of variables.*

A binding b of $t \in T$ maps variables $v \in \bigcup_{a \in A \cap ((P \times \{t\}) \cup (\{t\} \times P))} Var(\mathcal{E}(a)) \cup Var(Gd(t))$ to some value $b(v) \in Type(v)$. The set of all bindings of t is denoted by $B(t)$.

- A timed token element is a pair $((p,c),\iota)$, where $p \in P$ and $(c,\iota) \in cd(p)$.
- A binding element is a pair (t,b), where t is a transition and $b \in B(t)$.

We extend the bindings of t to guards, i.e. $b(Gd(t)) \in \{true, false\}$; to the untimed part of timed arc expressions, i.e. $b(e) \in \pi_1(cd(p))$ for all $e \in \mathcal{E}(p,t)$ and $(p,t) \in A \cap (P \times T)$ and $b(e) \in \pi_1(cd(p))$ for all $e \in \mathcal{E}(t,p)$ and $(t,p) \in A \cap (T \times P)$.

A marking is a bag of timed token elements. A step is a bag of binding elements.

A state of a TCPN $\mathcal{N}$ is a pair (M,ι), where M is a marking and ι is a time value called the creation time. The initial state is the pair $\mathfrak{s}_0 = (M_0, \iota_0)$.

Let M be a marking. We denote the bag over the color set of the place by $M(p) \stackrel{\text{def}}{=} \sum_{((p,c),\iota) \in M} M(((p,c),\iota))[(c,\iota)]$.

We define the transition relation which consists of two kinds of steps — untimed and timed[2]:

- A binding element (t,b) is enabled in a state (M,ι) at time ι iff $b(Gd(t)) = true$ and
$$\forall p \in P: \sum_{\substack{e \in \mathcal{E}(p,t) \\ (p,t) \in A \cap (P \times T)}} \mathcal{E}(p,t)(e)[b(e)^\iota] \leq M(p).$$
Then the binding element (t,b) can fire resulting in state (M',ι) and we write $(M,\iota) \xrightarrow{(t,b)} (M',\iota)$, so that for all $p \in P$
$$M'(p) = M(p) - \sum_{\substack{e \in \mathcal{E}(p,t) \\ (p,t) \in A \cap (P \times T)}} \mathcal{E}(p,t)(e)[b(e)^\iota] + \sum_{\substack{e \in \mathcal{E}(t,p) \\ (t,p) \in A \cap (T \times P)}} \mathcal{E}(t,p)(e)[b(e)^{\iota + e^t}].$$
- Let $\iota' > \iota$ be the smallest time at which a binding element (t,b) can become enabled in a state (M,ι). Then, a clock step takes place increasing the global time. We write $(M,\iota) \xrightarrow{\iota' - \iota} (M,\iota')$.

[2] We make the time step explicit in contrast with the semantics from [74] in order to be able to verify properties which take time into account.

Note that the above definitions hold when all places have a timed type. For the case when the color of a place p is untimed, we also write (p, c) for token elements and the definitions above hold by leaving time stamp out. In the sequel, we use both notations for untimed places.

We write $(M, \iota) \xrightarrow{(t,b)} (M', \iota')$, and $(M, \iota) \xrightarrow{t} (M', \iota')$ if there exists a binding $b \in B(t)$ such that $(M, \iota) \xrightarrow{(t,b)} (M', \iota')$. As usual, we extend this relation to sequences of binding elements and time values σ and write $(M, \iota) \xrightarrow{\sigma} (M', \iota')$ and $(M, \iota) \xrightarrow{*} (M', \iota')$ if there exists a sequence of binding elements and time values such that $(M, \iota) \xrightarrow{\sigma} (M', \iota')$. We denote the set of reachable states of $\mathcal{N}$ by $\mathfrak{S}(N, \mathfrak{s}_0) = \{\mathfrak{s} | \mathfrak{s}_0 \xrightarrow{*} \mathfrak{s}\}$. The labeled transition system of $\mathcal{N}$ is $\langle \mathfrak{S}(\mathcal{N}, \mathfrak{s}_0), \Sigma \cup \mathbb{T}, \longrightarrow, \mathfrak{s}_0 \rangle$.

2.6 Convex Geometry

Here we give some mathematical definitions and results we need (see e.g. [127] for more details). Let $\mathbb{D} \in \{\mathbb{N}, \mathbb{N}^+, \mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$ denote a generic numerical domain. For $n \in \mathbb{N}^+$, we denote by $\mathbb{D}^n$ the set of vectors with n components which take their values over $\mathbb{D}$.

Definition 2.35. [CONVEX SET]
A vector $u = \alpha_1 \cdot u_1 + \alpha_2 \cdot u_2 + \ldots + \alpha_k \cdot u_k$ where the α_i are nonnegative scalars from $\mathbb{D}$ such that $\sum_{i=1}^{k} \alpha_i = 1$ is called a convex combination *of the vectors $u_1, u_2, \ldots, u_k$ from $\mathbb{D}^n$.*

A set $Q \subseteq \mathbb{Q}^n$ is convex *iff for all pairs of vectors u_1, u_2 any convex combination of them is also in Q.*

Definition 2.36. [CONE]
A set of vectors $C \subseteq \mathbb{D}^n$ is called a cone *if, for every vector $u \in C$ and every nonnegative λ, $\lambda \cdot u \in C$.*

There are two equivalent definitions of convex polyhedral cones.

Definition 2.37. [CONVEX POLYHEDRAL CONE]

- **constraint representation** *A convex cone C is* polyhedral *if $C = \{x \mid A \cdot x \leq \bar{0}\}$ for some matrix $A \in \mathbb{D}^{n \times m}$, i.e. C is the intersection of finitely many linear half-spaces.*
- **generator representation** *A* convex polyhedral cone *C over $\mathbb{D}^n$ with a finite set of* generators *$E \subseteq \mathbb{D}^n$ is $C \stackrel{\text{def}}{=} \{\sum_{e \in E} \alpha_e \cdot e | \alpha_e \in \mathbb{D}^+\}$.*

A *trivial generator* is a vector $\bar{j}$, for some $1 \leq j \leq n$, such that $\bar{j}(j) = 1$ and $\bar{j}(j') = 0$ for all $j' \in \{1, \ldots, n\} \setminus \{j\}$.

For any two convex polyhedra $X, Y \in \mathbb{D}^n$, the *intersection* of X and Y is defined as the set intersection $X \cap Y$ and the *sum* of X and Y is defined as $X + Y \stackrel{\text{def}}{=} \{x + y \in \mathbb{D}^n | x \in X \wedge y \in Y\}$.

Lemma 2.3. *[127] Let X, Y be convex polyhedral cones. Then $X+Y$ and $X \cap Y$ are convex polyhedral cones.*

Computing the intersection of two polyhedra is easier if the polyhedra are given in the constraint representation, whereas the addition of two polyhedra is computed easier in the generator representation.

3

Generalized Soundness

We improve the decision procedure from [60] for the problem of generalized soundness of workflow nets. A workflow net is generalized sound iff every marking reachable from the initial marking with k tokens on the initial place terminates properly, i.e. it can reach a marking with k tokens on the final place, for an arbitrary natural number k. Our new decision procedure not only reports whether the net is sound or not, but also returns a counterexample in case the workflow net is not generalized sound. We report on experimental results obtained with the prototype we made and explain how the procedure can be used for the compositional verification of large workflows.

The chapter is based on [64].

3.1 Introduction

The traditional notion of soundness from [3] is not compositional [59]. For example if we refine the place d of the sound net N with the sound net N' from Figure 3, the obtained net (Figure 4) is not sound any more since $\bar{i} \xrightarrow{*} \bar{c} + \bar{e} + 2 \cdot \bar{i}' \xrightarrow{*} \bar{c} + \bar{e} + \bar{a}' + \bar{c}' + 2 \cdot \bar{b}' \xrightarrow{y} \bar{c} + \bar{e} + 2 \cdot \bar{b}' + \bar{f}' \not\xrightarrow{*} 2 \cdot \bar{f}'$.

A notion of *generalized soundness* was introduced in [59] and it amounts to the proper termination of all markings obtained from markings with multiple tokens on the initial place, which corresponds to the processing of batches of cases in the WF-net. It was proved that generalized soundness is compositional w.r.t. refinement, which allows the verification of soundness in a compositional way.

The generalized soundness problem is decidable and [60] gives a decision procedure for it. The problem of generalized soundness is reduced to two checks. First, some linear equations for the incidence matrix are checked. Secondly, the proper termination of a finite set of markings is checked. This finite set of markings is computed from an over-approximation of the set of reachable markings that has a regular algebraic structure. In practice, this set turns out to be very large, which seriously limits the applicability of the decision procedure given in [60].

In this chapter we show that the check of generalized soundness can be reduced to checking proper termination for a much smaller set of markings, namely minimal markings of the set from [60]. We describe a new decision procedure for soundness. Additionally, our procedure produces a counterexample in case a WF-net turns out to be unsound, showing a reachable marking that cannot terminate properly and a trace leading to it.

We implemented our decision procedure in a prototype tool and performed a series of experiments with it. The experimental results confirmed that the new procedure is considerably more effective than the old one. When applied together with standard reduction techniques in a compositional way, it allows us to check soundness of real-life models.

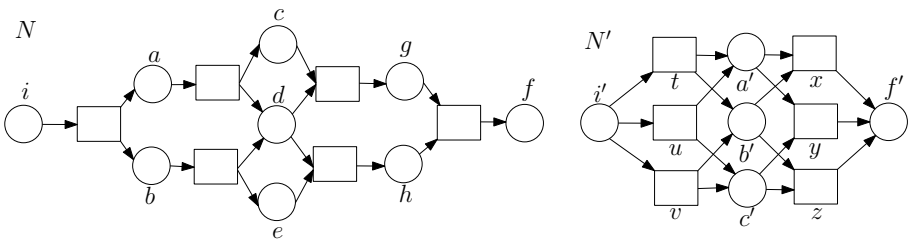


Fig. 3: Two sound nets

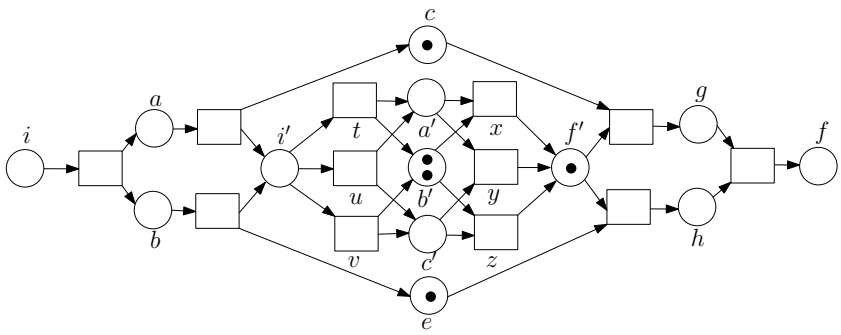


Fig. 4: The net obtained by refining place d of N with N' is not sound

Related work

For some subclasses of workflow nets (e.g. well-handled, free-choice, extended free-choice, asymmetric choice where every siphon includes at least one trap, extended non-self controlling workflow nets), 1-soundness implies generalized soundness (see [112]). A decidability proof for generalized soundness was presented in [132, 134], where the generalized soundness problem is reduced to the home marking problem in an extension of the workflow net, which is an *unbounded* net. The home marking problem is shown to be decidable in [42] by reducing it to the reachability problem for a finite set of markings. Checking generalized soundness in this way can however hardly be done in practice, since the complexity of the reachability problem for unbounded nets is still an open question, and the procedure for checking reachability, though known from 1981 [99], has never been implemented due to its complexity (the known algorithms require non-primitive recursive space) [119]. In our procedure we also check reachability for a finite set of markings, but reachability is checked on *bounded* nets only.

The chapter is structured as follows. Section 3.2 introduces basic notions. Section 3.3 presents the new decision procedure, and Section 3.4 provides details about the implementation and experimental results. Section 3.5 covers conclusions and directions for future work.

3.2 Soundness of Batch Workflow Nets

In [60], two structural correctness criteria for WF-nets based on siphons and traps were introduced:

Definition 3.1. [NON-REDUNDANCY, NON-PERSISTENCY]*[60]*
Let N be a workflow net.

- *A place $p \in P$ is* non-redundant *iff there exists $k \in \mathbb{N}$ and $m \in \mathbb{N}^P$ such that $k \cdot \bar{i} \xrightarrow{*} m \wedge m(p) > 0$.*

- *A transition $t \in T$ of N is called* non-redundant *if there exists a $k \in \mathbb{N}$ and $m \in \mathbb{N}^P$ such that $k \cdot \bar{i} \xrightarrow{*} m \xrightarrow{t}$. N is called* non-redundant *iff it does not contain any non-redundant place.*
- *A place $p \in P$ is* non-persistent *iff there exists $k \in \mathbb{N}$ and $m \in \mathbb{N}^P$ such that $m(p) > 0 \wedge m \xrightarrow{*} k \cdot \bar{f}$. N is called* non-persistent *iff it does not contain any non-persistent place.*

Non-redundancy means that every place can be marked and every transition can fire, provided there are enough tokens on the initial place, while non-persistency means that all places can become empty again. As proven in [60], non-redundancy and non-persistency are behavioral properties which imply restrictions on the structure of the net: all proper siphons of the net should contain i and all proper traps should contain f. If N contained a proper siphon without i, the output transitions for the places in the siphon would be dead due to Proposition 2.2.2. Similarly, if N contained a trap without f, the workflow could not reach the final marking with tokens in $[f]$, as all the tokens on the places of the trap remain within the trap (Proposition 2.2.1).

Following [60], we define a class of nets called batch workflow nets (BWF-nets).

Definition 3.2. [Batch workflow net]
A Batch Workflow net (BWF-net) *N is a Petri net having the following properties:*

1. *N has a single source place i and a single sink place f.*
2. *Every transition of N has at least one input and one output place.*
3. *Every proper siphon of N contains i.*
4. *Every proper trap of N contains f.*

It was shown in [60] that the last two conditions imply the path property of WF-nets in Definition 2.30, i.e. that every vertex is on the path from i to f. Actually, BWF-nets are WF-nets without redundant and persistent places, i.e. workflow nets that satisfy minimal correctness requirements.

Workflow nets were originally used to model the execution of one case. In [60], we defined a generalized notion of soundness for modeling the execution of batches of cases in WF-nets.

Definition 3.3. [k-soundness, generalized soundness]
A WF-net N is called k-sound *for some $k \in \mathbb{N}$ iff*

$$\mathcal{R}(N, k \cdot \bar{i}) \subseteq \mathcal{S}(N, k \cdot \bar{f}).$$

A WF-net N is called generalized sound *iff*

$$\forall k \in \mathbb{N}: \mathcal{R}(N, k \cdot \bar{i}) \subseteq \mathcal{S}(N, k \cdot \bar{f}).$$

For the sake of brevity, we omit the word "generalized" in the rest of this chapter. In [60], it has been shown that a WF-net N is sound iff a certain derived BWF-net N' is sound. The derivation is straightforward and only uses structural analysis of the net.

We further remind the procedure from [60] of obtaining BWF-nets from WF-nets that have the same behavior. Let N be a WF-net.

- Find a maximal siphon X in $P \setminus \{i\}$ using Algorithm 1. All places from X are redundant and the transitions from $X^\bullet$ are redundant as well. $(N', k \cdot \bar{i})$ obtained by removing places from X and transitions from $X^\bullet$ is either not a WF-net any more and so N was ill-designed, or it is a WF-net, which is an improved version of N.
- Check whether N' has persistent places by finding the largest trap X' in $P \setminus \{f\}$ using Algorithm 2. If $X' \neq \varnothing$, N' is not a sound WF-net. Otherwise, we can work with the BWF-net N' instead of N.

3.3 Decision Procedure

In this section, we describe our decision procedure for checking generalized soundness of BWF-nets. Our decision procedure improves the one from [60] since we check proper termination for a much smaller set of markings. We give an algorithm for computing this set of markings and enhance the procedure with a backward reachability algorithm that checks whether these markings are backward reachable from some final marking. If not, our procedure returns a counterexample.

We start by briefly discussing the decision procedure from [60]. We first give some necessary conditions of soundness:

Lemma 3.1. [60] *Let N be a sound BWF-net. Then,*

1. $\mathcal{I} \cdot \bar{i} = \mathcal{I} \cdot \bar{f}$ *(i and f agree on the basis place invariants);*
2. $\mathcal{I} \cdot x = \bar{0}$ *for* $x \in (\mathbb{Q}^+)^P$ *iff* $x = \bar{0}$.

The conditions of Lemma 3.1 can be easily checked by standard algebraic techniques. Further on, we consider only nets that meet these two conditions.

The set of all markings reachable from some initial marking of N is given by $\mathcal{R} = \bigcup_{k \in \mathbb{N}} \mathcal{R}(N, k \cdot \bar{i})$. Due to the marking equation, $\mathcal{R}(N, k \cdot \bar{i})$ is a subset of $\mathcal{G}_k = \{k \cdot \bar{i} + F \cdot v \mid v \in \mathbb{Z}^T\} \cap \mathbb{N}^P$. Note that the reverse is not true.

Let $m \in \mathcal{G}_k$, for some $k \in \mathbb{N}$. Then $\mathcal{I} \cdot m = \mathcal{I} \cdot (k \cdot \bar{i})$. Since condition 2 of Lemma 3.1 holds, $\mathcal{G}_k \cap \mathcal{G}_\ell = \varnothing$ for all $k, \ell \in \mathbb{N}$, with $k \neq l$, and we can define the *i-weight* function $w(m)$ for m as k. Now consider the set $\mathcal{G} = \bigcup_{k \in \mathbb{N}} \mathcal{G}_k$, i.e. $\mathcal{G} = \{k \cdot \bar{i} + F \cdot v \mid k \in \mathbb{N} \wedge v \in \mathbb{Z}^T\} \cap \mathbb{N}^P$. We will say that a marking $m \in \mathcal{G}$ *terminates properly* if $m \xrightarrow{*} w(m) \cdot \bar{f}$.

Lemma 3.2. [60] *Let $m_1, m_2 \in \mathcal{G}$ be markings that terminate properly and $m = \lambda_1 \cdot m_1 + \lambda_2 \cdot m_2$ for some $\lambda_1, \lambda_2 \in \mathbb{N}$. Then $m \in \mathcal{G}$ and it terminates properly.*

Theorem 3.1. [60] *Let N be a BWF-net. Then N is sound iff for any $m \in \mathcal{G}$, $m \xrightarrow{*} w(m) \cdot \bar{f}$.*

$\mathcal{G}$ is an infinite set, but unlike $\mathcal{R}$ it has a regular algebraic structure, which allows to reduce the check of proper termination to a check on a finite set of markings.

The following lemma is proved by using convexity analysis [127], notably the Farkas-Minkowski-Weyl theorem.

Lemma 3.3. [60] *Let $\mathcal{H} \stackrel{\text{def}}{=} \{a \cdot \bar{i} + F \cdot v \mid a \in \mathbb{Q}^+ \wedge v \in \mathbb{Q}^T\} \cap (\mathbb{Q}^+)^P$. Then there exist a finite set $E_{\mathcal{G}} \subseteq \mathcal{G}$ of generators of $\mathcal{H}$, i.e. $\mathcal{H} = \{\Sigma_{e \in E_{\mathcal{G}}} \lambda_e \cdot e \mid \lambda_e \in \mathbb{Q}^+\}$.*

The soundness check can now be reduced to the check of proper termination for a finite set of markings:

Theorem 3.2. [60] *Let N be a BWF-net such that the conditions of Lemma 3.1 hold and let $\Gamma \stackrel{\text{def}}{=} \{\sum_{e \in E_{\mathcal{G}}} \alpha_e \cdot e \mid 0 \leq \alpha_e \leq 1 \wedge e \in E_{\mathcal{G}}\} \cap \mathcal{G}$, where $E_{\mathcal{G}} \subseteq \mathcal{G}$ is a finite set of generators. Then N is sound iff all markings in Γ terminate properly.*

In fact, Γ represents the set of integer points of the bounded convex polyhedral cone (also called polytope) having the set $E_{\mathcal{G}}$ as generators.

The decision procedure from [60] comprises the following steps:

1. Find an invariant matrix $\mathcal{I}$ and check whether $\mathcal{I} \cdot \bar{i} = \mathcal{I} \cdot \bar{f}$ and whether $\mathcal{I} \cdot x = \bar{0}$ has only the trivial solution on $\mathbb{N}^P$;
2. Find a set $E_{\mathcal{G}} \subset \mathcal{G}$ of generators of $\mathcal{H}$;
3. Compute the set of markings Γ;
4. Check for all markings $m \in \Gamma$ that $m \xrightarrow{*} w(m) \cdot \bar{f}$.

Example 1. We illustrate the main steps of the algorithm on the BWF-net from Figure 5. First we compute $\mathcal{I} = (4, 1, 1, 4)$. The first two conditions are satisfied: $(4, 1, 1, 4) \cdot \bar{i} = (4, 1, 1, 4) \cdot \bar{f}$ and $(4, 1, 1, 4) \cdot x = \bar{0}$ implies $x = \bar{0}$. Further we compute $\mathcal{H} = \{a \cdot \bar{i} + F \cdot v | a \in \mathbb{Q}^+, v \in \mathbb{Q}^T\} \cap (\mathbb{Q}^+)^P = (A + B) \cap C$, where A, B and C are polyhedra having the generators $\{\bar{i}\}$, $\{\pm(3 \cdot \bar{a} + \bar{b} - \bar{i}), \pm(\bar{a} + \bar{b}), \pm(\bar{i} - \bar{a} - 3 \cdot \bar{b})\}$ and $\{\bar{i}, \bar{f}, \bar{a}, \bar{b}\}$, respectively. Next we compute the generators of the polytope: $E_{\mathcal{G}} = \{\bar{i}, \bar{f}, 8 \cdot \bar{a}, 8 \cdot \bar{b}\}$. The markings of Γ have the following form:

$$\Gamma = \{m' | (m' = \sum_{\substack{m \in E_{\mathcal{G}} \cup \{3 \cdot \bar{a} + \bar{b}, \bar{a} + 3 \cdot \bar{b}\} \\ \alpha_m \in \mathbb{N}}} \alpha_m \cdot m) \wedge (\bar{0} \leq m' \leq \sum_{e \in E_{\mathcal{G}}} e)\}$$

The size of Γ is very large compared to the size of the net: $|\Gamma| = 44$. Furthermore in order to check whether all markings of Γ terminate properly, we need to build the backward reachability sets $\mathcal{S}(N, k \cdot \bar{i})$ for $0 \leq k \leq \max_{m \in \Gamma} w(m) = 6$ and check whether they include all markings of Γ. We observe that $8 \cdot \bar{b} \notin \mathcal{S}(N, 2 \cdot \bar{f})$ and therefore the net is not sound.

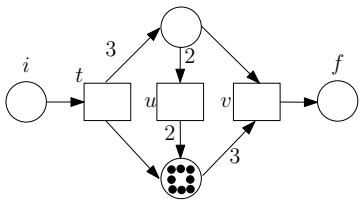


Fig. 5: Example of a BWF-net

Steps 1 – 2 are not computationally costly. The set of markings Γ turns out to be very large in practice. Step 3 and 4 are typically very expensive for real-life examples. We shall reduce the check of proper terminations of markings from Γ to a check of a smaller set of markings by replacing the last two steps with the following steps:

3'. Compute the set Υ of *minimal markings* of $\mathcal{G}^+ \stackrel{\text{def}}{=} \bigcup_{k \in \mathbb{N}^+} \mathcal{G}(k \cdot \bar{i})$, i.e.

$$\Upsilon \stackrel{\text{def}}{=} \{m \mid \forall m' \in \mathcal{G}^+ : m' \leq m \Rightarrow m' = m\}.$$

4'. Check that for all markings $m \in \Upsilon$, $m \xrightarrow{*} w(m) \cdot \bar{f}$. In case this does not hold, give a counterexample, i.e. a trace σ such that $w(m') \cdot \bar{i} \xrightarrow{\sigma} m' \not\xrightarrow{*} w(m') \cdot \bar{f}$, for some m'.

To show that Υ can be used instead of Γ, we first prove an auxiliary lemma.

Lemma 3.4. *Let N be a BWF-net, $m_1 \in \mathcal{G}_{k_1}$, $m_2 \in \mathcal{G}_{k_2}$, for some $k_1, k_2 \in \mathbb{N}$, and $m_2 > m_1$. Then $k_2 > k_1$.*

Proof. Since $m_2 \in \mathcal{G}_{k_2}$ and $m_1 \in \mathcal{G}_{k_1}$, $m_1 = k_1 \cdot \bar{i} + F \cdot v$ and $m_2 = k_2 \cdot \bar{i} + F \cdot v_2$ for some v_1 and v_2. Hence, $\mathcal{I} \cdot m_1 = \mathcal{I} \cdot k_1 \cdot \bar{i}$ and $\mathcal{I} \cdot m_2 = \mathcal{I} \cdot k_2 \cdot \bar{i}$ and by linearity $\mathcal{I}(m_2 - m_1 + (k_1 - k_2)\bar{i}) = \bar{0}$. Since condition 2 of Lemma 3.1 holds for N, $m_2 - m_1 + (k_1 - k_2) \cdot \bar{i} \notin (\mathbb{Q}^+)^P \setminus \{\bar{0}\}$. Since $m_2 - m_1$ and $\bar{i}$ are in $(\mathbb{Q}^+)^P$, we deduce that $k_1 - k_2 < 0$. □

The set of markings Υ has the following properties:

- Let $E_{\mathcal{G}} \subseteq \mathcal{G}$ be a set of minimal generators of $\mathcal{H}$ in $\mathcal{G}$ (i.e. for any $e \in E_{\mathcal{G}}$ and $e' \in \mathcal{G}$, $e' \leq e$ implies $e = e'$). Then $E_{\mathcal{G}} \subseteq \Upsilon$. Note that in particular $\bar{i}$, $\bar{f} \in E_{\mathcal{G}} \subseteq \Upsilon$.
- $\mathcal{G}_1 \subseteq \Upsilon$. Suppose that there is an $m \in \mathcal{G}_1$ such that $m \notin \Upsilon$. Then, there is $m' \in \Upsilon$ such that $m' < m$. By Lemma 3.4, $w(m') < w(m) = 1$, contradiction.

We now formulate our main theorem.

Theorem 3.3. *Let N be a BWF-net such that $\mathcal{I} \cdot \bar{i} = \mathcal{I} \cdot \bar{f}$, $\mathcal{I} \cdot x = \bar{0}$ has only the trivial solution in $(\mathbb{Q}^+)^P$, $\mathcal{G}^+ \stackrel{\text{def}}{=} \{k \cdot \bar{i} + F \cdot v \mid k \in \mathbb{N}^+ \wedge v \in \mathbb{Z}^T\} \cap \mathbb{N}^P$, $\mathcal{H} = \{a \cdot \bar{i} + F \cdot v \mid a \in \mathbb{Q}^+, v \in \mathbb{Q}^T\} \cap (\mathbb{Q}^+)^P$, $E_{\mathcal{G}} \subseteq \mathcal{G}^+$ be a set of minimal generators of $\mathcal{H}$ in $\mathcal{G}^+$, $\Gamma = \{\sum_{e \in E_{\mathcal{G}}} \alpha_e \cdot e \mid 0 \leq \alpha_e \leq 1 \wedge e \in E_{\mathcal{G}}\} \cap \mathcal{G}$, and Υ be the set of minimal markings of $\mathcal{G}^+$. Then:*

1. *N is sound iff for any marking $m \in \Upsilon$, $m \xrightarrow{*} w(m) \cdot \bar{f}$.*
2. *Each marking $m \in \Upsilon$ satisfies $m < M$, where $M(p) = \max_{e \in E_\mathcal{G}} e(p)$, for every $p \in P$.*
3. *$\Upsilon \subset \Gamma$.*

Proof. (1) ($\Rightarrow$) Since N is sound, all markings of $\mathcal{G}$ terminate properly (by Theorem 3.1). Since $\Upsilon \subseteq \mathcal{G}$, all markings of Υ terminate properly.
($\Leftarrow$) Let $m \xrightarrow{*} w(m) \cdot \bar{f}$ for every marking m from Υ. We will prove that $m \xrightarrow{*} w(m) \cdot \bar{f}$ for every marking m from Γ, which implies then that N is sound (by Theorem 3.2).

Let $m \in \Gamma$. We have two cases: $m \in \Upsilon$ and $m \in \Gamma \setminus \Upsilon$. If $m \in \Upsilon$, then $m \xrightarrow{*} w(m) \cdot \bar{f}$. If $m \in (\Gamma \setminus \Upsilon)$, $m > \Delta^0$ for some $\Delta^0 \in \Upsilon$ and by Lemma 3.4, $w(m) > w(\Delta^0)$, which also implies that $(m - \Delta^0) \in \mathcal{G}^+$.

Set $m^0 = m - \Delta^0$. In case $m^0 \in \Upsilon$, $m^0 \xrightarrow{*} w(m^0) \cdot \bar{f}$. By Lemma 3.2, since $\Delta^0 \xrightarrow{*} w(\Delta^0) \cdot \bar{f}$, $m \xrightarrow{*} w(m) \cdot \bar{f}$. In case $m^0 \notin \Upsilon$, m^0 can be further written as $m^0 = \Delta^1 + m^1$, where $\Delta^1 \in \Upsilon$ and $m^1 \in \mathcal{G}^+$.

We continue until we reach an $m^{l-1} = m^l + \Delta^l$ with $\Delta^l \in \Upsilon$ and $m^l \in \Upsilon$. Note that the process is finite since $0 < m^{i+1} < m^i$, for $0 \le i \le l$. Therefore $m = \sum_{i=0}^{l} \Delta^i + m^l$, where $m^l \in \Upsilon$ and $\Delta^i \in \Upsilon$ for all $i = 0 \ldots l$. Since the markings of Υ terminate properly, we can apply Lemma 3.2 to $\sum_{i=0}^{l} \Delta^l + m^0$. As a result, $m \xrightarrow{*} w(m) \cdot \bar{f}$.
(2) Suppose that there is a marking $m \in \Upsilon$ such that $m \ge M$. Since $M \ge e$ for every generator $e \in E_\mathcal{G}$, we have $\forall e \in E_\mathcal{G} : m \ge e$. That means that m and e are comparable, which contradicts the hypothesis.
(3) $\Upsilon \subseteq \Gamma$ follows trivially from (2) and the definition of Γ. Furthermore, $\bar{M} = \sum_{e \in E_\mathcal{G}} e \in \Gamma$. However $\bar{M} > M$ and from (2), we have that $\bar{M} \notin \Upsilon$, hence $\Upsilon \subset \Gamma$. □

Now we can describe the implementation of the steps $2, 3'$ and $4'$.

Computing generators of the convex polyhedral cone $\mathcal{H}$

$\mathcal{H}$ is given as the intersection of two polyhedra: A with the set of generators $\{\bar{i}\} \cup \{\pm F(t) \mid t \in T\}$ (column vectors of the matrices F and $-F$) and B with the set of generators $\{\bar{p} \mid p \in P\}$ (trivial generators). Let E be a (minimal) set of generators of the convex polyhedral cone $\mathcal{H} = \{a \cdot \bar{i} + F \cdot v \mid a \in \mathbb{Q}^+, v \in \mathbb{Q}^T\} \cap (\mathbb{Q}^+)^P$. All generators of $\mathcal{H}$ can be represented as $a \cdot \bar{i} + F \cdot v$, where $a \in \mathbb{Q}$ and $v \in \mathbb{Q}^T$ can be found by solving linear equations. In order to find the set of generators that are in $\mathcal{G}$ ($E_\mathcal{G}$), the generators of $\mathcal{H}$ need to be rescaled. The rescaling factor of each generator is the lcm of the denominators of a and v_t, for all $t \in T$ divided by the gcd of the numerators of them. $\bar{i}$ and $\bar{f}$ are generators of $\mathcal{H}$ with rescaling factor 1.

Computing Υ

The next step is to find Υ — the set of minimal markings of $\mathcal{G}$. Note that the markings of Υ are smaller than the marking M whose components are the maxima of the respective components of the rescaled generators (statement 2 of Theorem 3.3).

We compute Υ by an optimized enumeration of all vectors m from $\mathbb{N}^P$ which are smaller than M and checking whether $m = k \cdot \bar{i} + F \cdot v$ has a solution in $\mathbb{N}^+$, i.e. whether $m \in \mathcal{G}$. The optimization is due to avoiding the consideration of markings which are greater than some markings already added to Υ.

Checking proper termination for markings of Υ

We need to check whether $m \xrightarrow{*} w(m) \cdot \bar{f}$ for all $m \in \Upsilon$. Since condition 2 of Lemma 3.1 holds, we conclude that $\mathcal{S}(k \cdot \bar{f})$ is a finite set for any k. Therefore we employ a backward reachability algorithm to check proper termination of markings in Υ. Let J be the (finite) set of weights of markings from Υ. The backward reachability algorithm constructs for each i-weight $j \in J$, starting from weight 1, the backward reachability set B_j. We start from the marking $j \cdot \bar{f}$ and continue by adding markings $\{m - F_t \mid m \in B_j \wedge m - F_t^+ \geq \bar{0} \wedge t \in T\}$, where F_t is column of F corresponding to transition t, until B_j contains all markings from Υ_j or we reach the fixpoint $\mathcal{S}(N, j \cdot \bar{f})$. In the first case all markings of Υ_j terminate properly; as a result the BWF-net is sound. In the latter case the markings in Υ_j do not terminate properly; therefore the net is not sound. Note that the backward reachability sets B_j are distinct (since $\mathcal{G}_k \cap \mathcal{G}_\ell = \varnothing$ for any $k \neq \ell$).

This check results either in verdict "sound" (if all markings from Υ terminate properly), or "unsound" together with some marking that does not terminate properly otherwise.

Finding a counterexample

Let m be a marking from Υ_j returned by the check above as non-properly terminating. Like all markings from Υ_j, m does not necessarily belong to $\mathcal{R}(N, j \cdot \bar{i})$. To give a counterexample, we search through $\mathcal{R}(N, k \cdot \bar{i})$ $(k \geq w(m))$ to find a marking m' reachable from $w(m') \cdot \bar{i}$ and not terminating properly and show a trace σ such that $w(m') \cdot \bar{i} \xrightarrow{\sigma} m'$. There are few classes of nets for which the all solutions are reachable. Such a class are the equal-conflict systems [130].

Example 1 continued. We compute Υ for the example from Figure 5:

$$\Upsilon = \{\bar{i}, \bar{f}, 8 \cdot \bar{a}, 8 \cdot \bar{b}, \bar{a} + 3 \cdot \bar{b}, 3 \cdot \bar{a} + \bar{b}\}$$

Note that $|\Upsilon| = 6$, while $|\Gamma| = 44$. Moreover, the maximal i-weight of the markings of Υ is a lot smaller than that of the markings of Γ: $\max_{m \in \Upsilon} w(m) = 2 < \max_{m \in \Gamma} w(m) = 6$. Hence, we need to compute only $\mathcal{S}(N, \bar{f})$ and $\mathcal{S}(N, 2 \cdot \bar{f})$

Algorithm 3: Backward reachability check

Input: $N = (P, T, F)$, Υ, $J = \{w(m) \mid m \in \Upsilon\}$
Output: "the BWF-net is sound" or "the BWF-net is not sound, m, k" where $m \in \mathcal{G}_k$ and $m \overset{*}{\not\rightarrow} k \cdot \bar{f}$.

for $j \in J$ **do**
  $B_j = \{j \cdot \bar{f}\}$;
  repeat
    $B_j = B_j \cup \{m - F_t \mid \forall p \in P : m(p) \geq F(p,t) \wedge m \in B_j \wedge t \in T\}$
  until *a fixpoint is reached or* $\Upsilon_j \subseteq B_j$;
  if $\Upsilon_j \nsubseteq B_j$ **then**
    pick $m \in \Upsilon_j \setminus B_j$;
    return ("the BWF-net is not sound", m, j)
  end
end
return ("the BWF-net is sound")

instead of $\mathcal{S}(N, k \cdot \bar{f})$ for $k = 1 \ldots 6$. We find a counterexample $8 \cdot \bar{b} \in \mathcal{R}(N, 2 \cdot \bar{i})$: $2 \cdot \bar{i} \xrightarrow{tt} 6 \cdot \bar{a} + 2 \cdot \bar{b} \xrightarrow{uuu} 8 \cdot \bar{b}$ and conclude that the net is not sound. Figure 5 shows the dead marking.

Example 2. The net N' from Figure 3 shows a Petri net which is 1-sound, but not 2-sound. In this case $\Upsilon = \Upsilon_1 = \{\bar{i}, \bar{f}, \bar{a}, \bar{b}, \bar{c}\} = E_{\mathcal{G}}$. Using the backward reachability algorithm, we find that the net is not sound and $\bar{b} \in \Upsilon_1$ such that $\bar{b} \overset{*}{\not\rightarrow} \bar{f}$. However, $\bar{b} \notin \mathcal{R}(N, \bar{i})$. We find $2 \cdot \bar{b} + \bar{f} > \bar{b}$ such that $2 \cdot \bar{i} \xrightarrow{tvy} 2 \cdot \bar{b} + \bar{f} \overset{*}{\not\rightarrow} 2 \cdot \bar{f}$.

3.4 Implementation of the Decision Procedure

In this section, we discuss how to check soundness for large nets compositionally by using reduction techniques and give some details on the implementation of the procedure and experimental results.

Using reduction rules to verify soundness

We can apply our procedure in combination with reduction techniques that preserve soundness in order to reduce the size of the net for which we are checking soundness.

We start with introducing the notion of *k-closure of a BWF-net*, which is the strongly connected net obtained by adding a transition whose only input place is the final place, the only output place is the initial place, and the weights of the arcs equal k.

Definition 3.4. [k-CLOSURE]
The k-closure of a WF-net $N = \langle P, T, F^+, F^- \rangle$ *is a net* $\langle P, T \cup \{\bar{t}\}, \bar{F}^+, \bar{F}^- \rangle$,

where $\bar{F}^-(i,\bar{t}) = \bar{F}^+(f,\bar{t}) = 0$, $\bar{F}^+(i,\bar{t}) = \bar{F}^-(f,\bar{t}) = k$, $\bar{F}^+(p,t) = F^+(p,t)$ *and* $\bar{F}^-(p,t) = F^-(p,t)$ *for all* $(p,t) \in P \times T$.

Lemma 3.5. *The k-closure of a BWF-net N is bounded and* $\bar{t}$*-live iff N is k-sound.*

Proof. ($\Rightarrow$) Since the closure of N is $\bar{t}$-live, for all $m \in \mathcal{R}(N, k \cdot \bar{i})$, there exists an m' such that $m \xrightarrow{*} m' \xrightarrow{\bar{t}} m''$. Boundedness of N implies $m' = k \cdot \bar{f}$ and $m'' = k \cdot \bar{i}$. Thus, N is sound.

($\Leftarrow$) Suppose the closure of N is unbounded. Then there exists $m \in \mathcal{R}(N, k\cdot\bar{i})$ such that $m \xrightarrow{*} m'$ and $m < m'$. Since N is k-sound, $m \xrightarrow{*} k \cdot \bar{f}$ and $m' \xrightarrow{*} k \cdot \bar{f} + m - m'$, which contradicts soundness of N. Hence N is bounded. By k-soundness of N, for all $m \in \mathcal{R}(N, k \cdot \bar{i})$, $m \xrightarrow{*} k \cdot \bar{f}$. Hence, $m \xrightarrow{*} k \cdot \bar{f} \xrightarrow{\bar{t}} k \cdot \bar{i}$. Thus the closure of N is $\bar{t}$-live. □

Thus, natural candidates for preserving soundness are rules that preserve $\bar{t}$-liveness and boundedness of the closure of the net in both directions, i.e. the closure of the BWF-net is $\bar{t}$-live and bounded iff the reduced closure of the BWF-net is $\bar{t}$-live and bounded. Such rules have been intensively investigated; among them, we recall the place substitution rule and the identical transitions rule of Berthelot [22] and the reduction rules Murata [104] (fusion of series places/transitions, fusion of parallel places/transitions, elimination of self loop transitions).

Let $\mathfrak{R}$ be a set of transformation rules between two k-closures of a BWF-net which preserve boundedness and $\bar{t}$-liveness in both directions (we also assume that $\bar{t}$, i and f are not reduced). Note that since the only initially marked place is i, the transformations from $\mathfrak{R}$ are applied to unmarked places only.

Soundness is preserved by applying rules from $\mathfrak{R}$ to the closure a BWF-net:

Theorem 3.4. *A BWF-net is sound iff the BWF-net obtained by applying reductions from the set* $\mathfrak{R}$ *is sound.*

Proof. By Lemma 3.5 soundness of a BWF-net is equivalent to the boundedness and $\bar{t}$-liveness of the k-closure of the BWF-net, for all $k \in \mathbb{N}$. The latter is equivalent to the boundedness and $\bar{t}$-liveness of the k-closure of the reduced BWF-net, for all $k \in \mathbb{N}$. By applying again Lemma 3.5, this is equivalent to the soundness of the reduced BWF-net. □

Compositional verification of soundness

In practice it is often needed to verify soundness of large workflow nets that cannot be handled by current verification tools. Therefore, a more efficient approach is needed to handle these cases. Applying simple reduction rules that preserve soundness, like the ones from [104], facilitates the task a lot. The reduced net can then be checked using a compositional approach:

1. Identify BWF-subnets in the original workflow by using classical graph techniques (e.g. by detecting strongly connected components).
2. Check whether the found BWF-subnets are generalized sound using the procedure described in Section 3.3.
3. Reduce every sound BWF-subnet to one place and repeat the procedure iteratively, till the soundness of the whole net is determined.

Correctness of the reduction part of Step 3 in the procedure above is justified by Theorem 6 from [59].

The procedure above has been has been implemented and integrated in the Petri net editor Yasper [144], which now supports arbitrary on-the-fly reductions (see [141] for more details) and uses the prototype check for generalized soundness which we will describe in the next paragraph.

Implementation and experimental results

In this section we focus on the implementation of the decision procedure described in Section 3.3. The tool uses the Yasper editor [63, 144] for input of batch workflow nets and gives as output whether the net is sound and a counterexample in case the net is not sound. The prototype is written in C++ and uses the Parma Polyhedra Library [18, 20] for the computation of the minimal set of generators of the convex polyhedral cone $\mathcal{H}$.

The complexity of the procedure is dominated by the complexity of the reachability problem; however, for BWF-nets modeling real-life business processes the performance turned out to be acceptable. We have run our prototype on a series of examples. The nets were first reduced with standard reduction rules from [104], which preserve soundness. Table 1 shows the experimental results comparing the size of Γ with the size of Υ. In most of the experiments Υ turned out to be equal to the set of rescaled generators. Our experiments showed that our tool can handle models of business processes of realistic size in reasonable time; a typical case: for a (reduced) BWF-net with $|P| = 18$ and $|T| = 22$, our algorithm checks soundness within 8 seconds.

It can be seen that the number of reachability checks can be improved with a factor 57, even when the nets are of relatively small size ($P = 9/T = 10$). For our examples, the size of Γ dominates the size of Υ on an average of 17 times. The size of Υ turns out to be generally polynomial in size of the net. Moreover, the weight of the markings in Υ is typically 1, which means that most of the time the backward reachability check needed to be performed only once (starting from $\bar{f}$), whereas for γ the weight of the markings is extremely high (up to 75).

3.5 Conclusion

In this chapter, we have presented an improved procedure for deciding generalized soundness of BWF-nets. We showed that the problem can be reduced to

Net	Soundness	$\|P\|$	$\|T\|$	$\|\Gamma\|/\|\Upsilon\|$	$\max_{m \epsilon \Gamma} w(m)$	$\max_{m \epsilon \Upsilon} w(m)$	$\|\Upsilon\|$	Time(ms)
1	sound	23	27	19	75	1	75 (= $\|E_{\mathcal{G}}\|$)	19909
2	sound	18	22	11	70	1	70 (= $\|E_{\mathcal{G}}\|$)	8005
3	sound	12	12	46	14	1	14 (= $\|E_{\mathcal{G}}\|$)	131
4	sound	11	12	19	5	1	43	133
5	sound	11	8	12	7	1	16	334
6	sound	9	10	57	9	1	9 (= $\|E_{\mathcal{G}}\|$)	16
7	sound	9	9	18	10	1	10 (= $\|E_{\mathcal{G}}\|$)	26
8	sound	7	8	18	8	2	7 (= $\|E_{\mathcal{G}}\|$)	9
9	sound	9	6	8	11	1	11 (= $\|E_{\mathcal{G}}\|$)	48
10	sound	6	6	6	6	1	6 (= $\|E_{\mathcal{G}}\|$)	9
11	sound	7	5	5	6	1	6 (= $\|E_{\mathcal{G}}\|$)	5
12	sound	6	5	6	4	1	6	8
13	not 2-sound	5	6	6	5	1	5 (= $\|E_{\mathcal{G}}\|$)	5
14	sound	5	5	8	5	1	5	7
15	not 2-sound	4	3	7	6	2	6	8

Table 1: Experimental results

checking proper termination for a set of *minimal markings* from the set found in [60], which significantly reduces the number of markings for which proper termination has to be checked. Further, we described a backwards reachability algorithm for checking proper termination for the found set of markings.

As discussed in Section 3.4, soundness of workflow nets can be checked in a compositional way. In addition to that, our soundness check can be used for compositional verification of Petri net properties. By adapting the proof of Theorem 6 from [59], it is easy to prove that if a Petri net has a subnet which is a generalized sound net whose transitions are labelled by invisible labels, the net obtained by reducing this subnet to one place is branching bisimilar to the original net.

4

Adaptive Workflow Nets

In this chapter, we consider Adaptive Workflow Nets, a subclass of nested nets that allows for more comfort and expressive power for modeling adaptation and exception handling than hierarchical nets. Flexibility is achieved by creating new nets out of existing ones and by allowing to change net structure at runtime. A typical domain with a great need for this kind of workflows is healthcare. Our running example is inspired by challenges in this domain.

We define two important behavioral properties of adaptive workflow nets: Soundness and Circumspectness. Soundness means that a proper final marking (state) can be reached from any marking which is reachable from the initial marking, and no garbage will be left. Circumspectness means that the upper layer is always ready to handle any decision or exception triggered at a lower layer. We employ an abstraction to reduce the problem of verifying soundness and circumspectness to a finite one. We show that even for adaptive workflow nets with infinite state space, these properties can be analyzed.

The chapter is based on [62, 68]. A previous version of [68] appeared as [67].

4.1 Introduction

In this chapter we consider *adaptive workflows*. In classical workflow management systems the process structure is determined at design time. During execution no structural changes are possible. This implies that designers need to take into account all possible executions, exceptional situations and combinations of them. In case of so-called ad hoc workflows [5, 6, 52, 120] the algorithm for processing cases is not known at design time, so it is impossible to use a classical workflow management system and so-called case handling systems are used instead. These systems have no formal process semantics which makes testing and verification impossible.

We propose *adaptive workflow systems* [62] as a solution with more flexibility for adaptation than classical workflow systems and more structure than ad hoc workflow systems. We assume a given library of protocols to be used as basic building blocks for constructing complex protocols. Then we model complex protocols in an adaptive way by using nested Petri nets [93, 94, 95]. Nested nets are an instance of "nets in nets" paradigm [136], where tokens in a (higher-level) net, called a *system net*, can be nets themselves, called *token nets*. Firings of the system net can depend on the synchronization with firings of the token nets. We allow a token net to be destroyed (made unusable) at an arbitrary marking that is determined at runtime. This situation cannot be easily modeled by hierarchical Petri nets, while inhibitor and reset arcs would make the model unreadable. Moreover, tokens in adaptive workflow systems can be modified in a structured way, for instance by extending it with another process from the library or by modifying the library of protocols at runtime.

As explained above, the adaptivity comes from the ability of the process to modify itself. Our framework is flexible enough to incorporate adaptability, which is the property of a process to be modified by an external party. In such a setting, the owner of a process would be able to use a breakthrough procedure, more reliable than existent from the library as long as the expected results are the same.

Using adaptive workflow systems one can achieve separation of concerns in modeling processes for different user groups. For instance, a doctor can follow a medical guideline that prescribes a blood test to be performed (system net) but he does not need to know how exactly the test is performed in the lab (token net). The further course of doctor's actions will however depend on the test results (synchronization).

Exception handling is an important feature in adaptive systems. In [14], the authors suggested to extend workflow systems with an exception handling mechanism that would allow to reason about the robustness (ability of the system to recover from error conditions) of such workflow. However, in their case no formal reasoning involving exceptions is possible. We extend the traditional workflow paradigm by introducing a mechanism for exception handling [62]. In our exception workflow nets we introduce exceptional transitions whose firings reflect exceptional situations.

We defined in [62] a non-recursive extension of nested nets [94] that have the same power as Turing machines, extended synchronization mechanism (in comparison to [94]) and furthermore special operations on (token) nets defined by expressions on arcs. Adaptive nets [62] were derived from this class, by restricting the structure of nested workflow nets (exceptional paths are reduced arcs between places of the workflow and exception transitions) and by restricting the coloring of the net, i.e. the expressions and guards allowed were proper termination and exception guards. In [68], we defined a subclass of adaptive nets from [62], by restricting the coloring of a net to black tokens and net tokens and allowing exception transitions to be connected only to places that belong to paths from the initial place to the final place. Adaptive workflow nets provide modeling comfort and are expressive enough to model many adaptive workflow applications and medical protocols.

We study two correctness properties of adaptive workflow nets: *soundness* and *circumspectness*. Similar to classical workflow nets, an adaptive workflow net is sound iff from any marking reachable from the initial marking (without firings of exception transitions) it is possible to reach the final marking (without firings of exception transitions or synchronizing on exceptions). Circumspectness means that any exception or final transition enabled in a token net of an adaptive workflow net can synchronize with a corresponding transition of the system net, i.e. any exception/decision in a lower layer can be handled by the upper layer of the system.

We identify subclasses of adaptive nets for which soundness and circumspectness is decidable and show how to reduce the verification of soundness and circumspectness of these adaptive nets (with possibly an infinite state space) to the verification of properties of their finite abstractions. The abstraction we use maps a token net to the set of exceptions specified in this net. We give a necessary and sufficient condition for soundness and circumspectness formulated in terms of the behavior of abstracted nets. This allows us to verify this nets in a compositional manner, by verifying an abstraction of every net token net and of the system net in the model.

The remainder of the chapter is organized as follows. Section 4.2 describes the running example as a motivation for introducing adaptive nets. In Section 4.3, we give a formalization of adaptive nets and in Section 4.4, we give some verification results. Section 4.6 concludes the chapter.

4.2 Example: a Medical Protocol

Nowadays medical protocols have the form of *guidelines*. There have been several attempts to formalize guidelines as *flowcharts* and *decision diagrams* and incorporate them into medical decision support systems (MDSS). For example, GLIF [108] and GUIDE [114] support medical processes by enacting guidelines.

As a motivating example we consider the process of diagnosis and treatment of a small-cell lung cancer (SCLC). The example is inspired by the medical

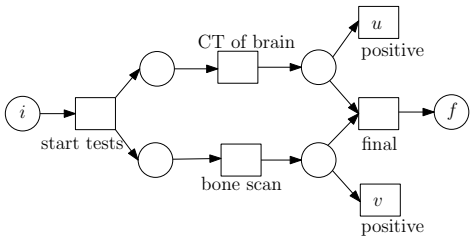


Fig. 6: Mandatory test protocol ($MandT\langle positive\rangle_e$)

guideline for SCLC [53] created by the National Comprehensive Cancer Network (SUA) and modified by the University of Texas M.D. Anderson Cancer Center for their patient population.

The guideline is built on the basis of a library of standard protocols. A protocol may describe the decision process for establishing a diagnosis, treatment schemes and tests used in the process of diagnosis or after the treatment. Each protocol has an initial point, a final point and it handles exceptional situations by terminating the process. For instance, the protocol used for mandatory tests in the initial diagnosis stage describes the process of executing two tests (computer tomography of brain and the bone scan) and evaluating their results. An exceptional situation is considered when the result of at least one of the tests turns out positive. We model such a protocol as an *exception workflow net*, i.e. a workflow net extended with special *exception* transitions, which are transitions without output places. These transitions carry an exception label indicating an exceptional situation. Figure 6 shows an extended workflow net modeling the mandatory test protocol; u and v are exception transitions with label *positive*.

The interfaces of protocols in the library are given as $Prot\langle\Sigma_e\rangle_e$, where $Prot$ is the name of the protocol and Σ_e are the sets of exception labels.

Figure 7 shows another protocol net ($STests\langle partial\ response,\ relapse\rangle_e$), modeling the test protocol used in the surveillance stage with two exception labels *partial response* and *relapse*. The process performs some tests such as CXR test, creatine test and liver function test. The progress of the cancer and the response to the treatment are evaluated: the cancer either relapses (modeled by the transition labeled *relapse*) or the treatment has a partial response (modeled by transition labeled accordingly).

Protocols can make use of other protocols, create new protocols from the existing ones or modify them. For this purpose, we consider nets as colored tokens and call such protocols *extended workflow nets*. The protocols can be built of more primitive ones by using the operations '.', '$\|$' and '+'. The operation *init* initializes an extended workflow net with its initial marking $[i]$.

Consider the surveillance protocol in Figure 8. The protocol makes use of other therapy and test protocols, namely *Radiotherapy*, $STests\langle partial\ re-$

sponse, *relapse*⟩$_e$, *Cisplatin*, *Etoposide*, *Prophylactic* and *RadControl*⟨*radiation scarring*⟩$_e$. The protocol iterates the surveillance treatment scheme until the results of the surveillance tests show signs of relapse. A regular surveillance treatment is started at runtime by creating a process token (denoted by the constant str = *init*(*Radiotherapy*. *STests*⟨*partial response*, *relapse*⟩$_e$) on the outgoing arc of the transition *start surveillance*). The process of this token consists of the protocol *Radiotherapy* followed by the *STests*⟨*partial response*, *relapse*⟩$_e$ protocol. The transition labeled by *partial response* in the token net located in place p is synchronized with a transition having the same label in the upper level net (*Surveillance*).

In case of a partial response, a new net token is created, namely ((CI. ET)‖*Radiotherapy*). ST , [i]) (radiotherapy performed in parallel with chemotherapy — cisplatin treatment followed by an etoposide treatment — followed by the ST protocol).

Additionally to the application of a surveillance treatment, a radiation control protocol (*RadControl*⟨*radiation scarring*⟩$_e$) is used to monitor the radiation effect on the patient. In case the patient shows signs of scars due to the radiation (the exception *radiation scarring* is signaled), a prophylactic treatment (*Prophylactic*) is conducted in parallel with the actual surveillance protocol (*st*‖*init*(*Prophylactic*)).

The protocol from the library describing the main process is depicted in Figure 9. The process starts with the decision whether the patient has small-cell lung cancer (SCLC) or non-small cell lung cancer (Non-SCLC). The transition *Non-SCLC* models an exception that allows finishing the guideline.

Once the diagnosis has been established, the stage of the illness needs to be assessed. The stage is determined by the extent of spread of the cancer basing on the test results. Here, the protocol makes use of and combines some standard test procedures depending on the preliminary diagnosis of the phase of the cancer. In case the patient shows signs of the extensive stage, a procedure with mandatory tests *MandT*⟨*positive*⟩$_e$ (Figure 6) is created. Once the

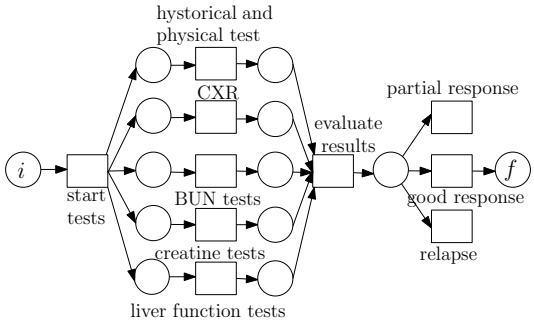
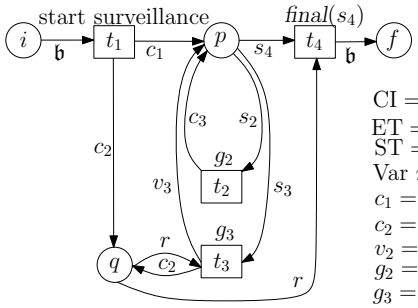
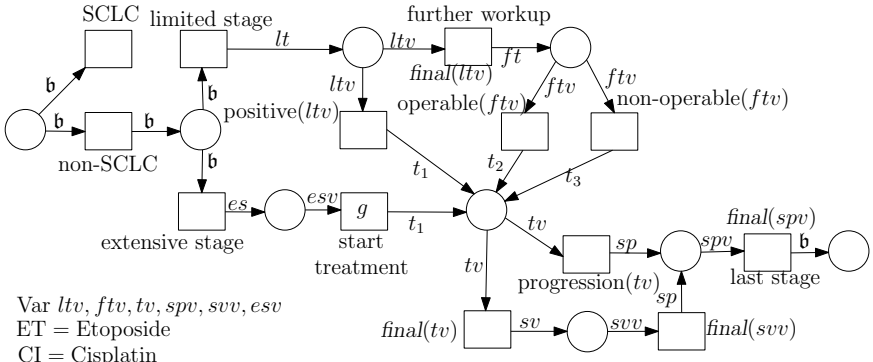


Fig. 7: Surveillance test protocol (*STests*⟨*partial response*, *relapse*⟩$_e$)



CI = Cisplatin
ET = Etoposide
ST = STests⟨partial response, relapse⟩$_e$
Var r, s_2, s_3, s_4
c_1 = *init*(Radiotherapy.ST)
c_2 = *init*(RadCon⟨radiation scarring⟩$_e$)
v_2 = *init*(((*CI.ET* ‖ Radiotherapy.ST)
g_2 = partial response(s_2)
g_3 = radiation scarring(r)
v_3 = s_3 ‖ *init*(Prophylactic)

Fig. 8: *Surveillance*



Var $ltv, ftv, tv, spv, svv, esv$
ET = Etoposide
CI = Cisplatin
sv = *init*(Surveillance)
sp = *init*(SalvagePalliation)
es = *init*(MandT⟨positive⟩$_e$)
ft = *init*(FTests⟨operable, non-operable⟩$_e$)
lt = *init*(MandT⟨positive⟩$_e$) ‖ FacT⟨positive⟩$_e$)
t_1 = *init*((Cisplatin. Etoposide. Tests⟨progression⟩$_e$)^6)
t_2 = *init*(Resection⟨final⟩$_f$⟨∅⟩$_e$. (*CI. ET*. Tests⟨progression⟩$_e$)^4)
t_3 = *init*(((Radiotherapy ‖ (*CI. ET*)). Tests⟨progression⟩$_e$)^4)
g = *final*(esv)

Fig. 9: Main SCLC protocol

protocol *MandT*⟨*positive*⟩$_e$ has terminated (tests are negative) or a transition with exception label *positive* (indicating that one of the tests is positive) has fired (the guard $ext = positive(esv)$ is true), a specific treatment protocol can be started.

In case the preliminary diagnosis shows signs of the limited stage (*limited stage* transition) more tests are needed than in the extensive case. The protocol executing these tests combines two existing test protocols taken from the library, i.e. *MandT*⟨*positive*⟩$_e$ and *FacT*⟨*positive*⟩$_e$, by performing them in parallel. Once one of the tests has a positive outcome, the transition labeled

positive fires, synchronized with the respective transition in the main protocol since this is a symptom for the extensive stage cancer. If none of the results turns out positive, the test protocol terminates properly.

The patient with limited-stage SCLC is further tested to determine whether or not she/he can be operated. For this purpose, the protocol $FTests\langle operable, non\text{-}operable\rangle_e$ is instantiated, and one of the exceptional outcomes of this test procedure (the patient is non-operable or operable) is synchronized with the respective transition in the SCLC protocol.

For each of the three diagnosis (extensive stage, limited stage operable, and limited stage non-operable), a special treatment scheme is created, which is actually an iteration of chemotherapy, radiotherapy and tests. For example, for the limited stage operable diagnosis, a treatment scheme considering the resection (*Resection*), followed by four iterations of chemotherapy (*Cisplatin* treatment followed by *Etoposide* treatment) and tests ($Tests\langle progression\rangle_e$) is created. Once there is a sign of progression of the cancer after performing the tests at each of the cycles signaled by the occurrence of an exception handled by the transition labeled by *progression*), the treatment is interrupted and the patient goes to the final stage, where only the Palliative/Salvage treatment can be applied. In case the initial treatment has been successfully completed (the transition labeled by *last stage* is fired) a surveillance protocol (*Surveillance*) is created and its completion determines the patient to enter the palliation/salvage stage.

4.3 Adaptive Workflow Nets

In this section, we introduce workflow nets extended with exceptions and then adaptive workflow nets, which are a special type of nested nets based on exception workflow nets.

4.3.1 Exception Workflow Nets

We extend the notion of a WF net to model *exceptions*. Exceptional situations occurring in the workflow often lead to the termination of the normal course of activities and reporting the exception. We model the reporting of exceptions by transitions labeled with exception labels whose execution terminates the execution of the WF net.

We consider a partition of the set of transitions $T = T_e \cup T_n$, where T_e is the set of *exception transitions*, T_n is the set of *normal transitions*. The set Σ of labels is partitioned into $\Sigma_e \cup \Sigma_n$ accordingly.

Definition 4.1. [Projection]
Let $N = \langle P, T, A, l\rangle$ *be a Petri net and* $T' \subseteq T$. *The* projection $N_{|T'}$ of N w.r.t. T' *is a net* $\langle P, T', A', l_{|T'}\rangle$, *where* $A' = ((P \times T') \cup (T' \times P)) \cap A$.

Definition 4.2. [Exception workflow net]
A Petri net $N = \langle P, T_e \cup T_n, A, l \rangle$ *is an* exception workflow net (EWF net) *iff the following conditions hold:*

1. *The net* $N_{|T_n}$ *is a workflow net.*
2. *For all* $t \in T_e$, $t^\bullet = \varnothing$, $^\bullet t \neq \varnothing$ *and* $l(t) \in \Sigma_e$.
3. *For all* $t \in T_n$ *and* $l(t) \in \Sigma_n$.

Note that WF nets are EWF nets with the empty set of exception transitions. The protocols $MandT\langle positive \rangle_e$ (Figure 6) and $STests\langle partial\ response, relapse \rangle_e$(Figure 7) are EWF nets.

Like for classical WF nets [9], an important correctness property for EWF nets is *soundness*, which requires that the process is always able to terminate properly by reaching the final marking and that all transitions can eventually fire, i.e. the net does not contain redundant transitions. We adapt this definition by requiring the possibility to properly terminate without using exception transitions for all markings reached without the use of exception transitions and by emphasizing the firing of a final transition.

Definition 4.3. [Soundness of EWF nets]
An EWF net N *is* sound *iff*

1. *(proper termination) for all* M *such that* $[i] \xrightarrow{\sigma} M$ *for some* $\sigma \in T_n^*$, $M \xrightarrow{\sigma'} [f]$ *for some* $\sigma' \in T_n^*$, *and*
2. $(N, [i])$ *is quasi-live.*

Definition 4.3 does not impose the condition stating that whenever the final state is reached, the execution has been completed. This condition of soundness is redundant both for the classical WF nets (cf. [60]) and for the EWF nets defined here:

Lemma 4.1. *Let* N *be a sound EWF net and* $[i] \xrightarrow{\sigma} M + [f]$ *for some* $\sigma \in T_n^*$. *Then* $M = \varnothing$.

Proof. Let $[i] \xrightarrow{\sigma} M + [f]$ where $\sigma \in T_n^*$. Since N is sound, there exists a $\sigma' \in T_n^*$ such that $M + [f] \xrightarrow{\sigma'} [f]$. Since $f^\bullet = \varnothing$, $M \xrightarrow{\sigma'} \varnothing$. For every transition t from σ', $t^\bullet \neq \varnothing$. Hence, $\sigma' = \epsilon$ and thus $M = \varnothing$. □

Proposition 4.1. *An EWF net* $(N, [i])$ *is quasi-live iff for every* $t \in T_e \cup T_n$, *there is a marking* $M \in \mathcal{R}(N_{|T_n}, [i])$ *such that* $^\bullet t \leq M$.

Proof. ($\Rightarrow$) Let $t_q \in T_n \cup T_e$. Since t_q is quasi-live in $(N, [i])$, $[i] \xrightarrow{\sigma} M \xrightarrow{t_q}$ for some M in $(N, [i])$.

First we prove that $[i] \xrightarrow{\sigma_{|T_n}} M'$ in N and $M \leq M'$ by induction on the length of σ. If $\sigma = \epsilon$, then $[i] = M = M'$. Suppose that the statement holds for all $\sigma \in (T_e \cup T_n)^k$, for some $k \in \mathbb{N}$. Let $[i] \xrightarrow{\sigma} M_1 \xrightarrow{t} M$ for some $\sigma t \in (T_n \cup T_e)^{k+1}$.

By the induction hypothesis, we have $[i] \xrightarrow{\sigma_{|T_n}} M_1'$ and $M_1 \leq M_1'$. If $t \in T_n$, $M_1 \xrightarrow{t} M$ and $M_1 \leq M_1'$ imply $M_1' \xrightarrow{t} M'$ in $N_{|T_n}$ and $M \leq M'$. If $t \in T_e$, ${}^\bullet t \not\in \varnothing$ and $t^\bullet = \varnothing$. Hence $M_1 > M$. Since $(\sigma t)_{|T_n} = \sigma_{|T_n}$, $[i] \xrightarrow{(\sigma t)_{|T_n}} M'$ and $M < M'$.

Hence ${}^\bullet t_q \leq M \leq M'$, thus t_q is quasi-live in $(N_{|T_n}, [i])$.

($\Leftarrow$) Trivially follows from the definition of the firing relation. □

Proposition 4.2. *Let N be an EWF net $N = \langle P, T_e \cup T_n, F, l \rangle$. Then condition* (2) *of soundness holds for N iff it holds for $N_{|T_n}$.*

Proof. ($\Rightarrow$) Let $M \in \mathcal{R}(N_{|T_n}, [i])$, i.e. $[i] \xrightarrow{\sigma} M$ in $N_{|T_n}$ for $\sigma \in T_n^*$. Hence $[i] \xrightarrow{\sigma} M$ in N. Since N is sound, $M \xrightarrow{\sigma'} [f]$ for some $\sigma' \in T_n^*$, which implies $M \xrightarrow{\sigma'} [f]$ in $N_{|T_n}$.

($\Leftarrow$) Let $[i] \xrightarrow{\sigma} M$ in N for $\sigma \in T_n^*$. Then $M \in \mathcal{R}(N_{|T_n}, [i])$ and $M \xrightarrow{\sigma'} [f]$ in $N_{|T_n}$ for some $\sigma' \in T_n^*$. Then $M \xrightarrow{\sigma'} [f]$ in N. □

Note that soundness of an EWF net N implies that its projection $N_{|T_n}$ is sound but not the other way around (because of the quasi-liveness requirement that also implies the requirement of quasi-liveness on exception transitions). Since every sound WF net is bounded [9], sound EWF nets are bounded. Furthermore, quasi-liveness of the exception transitions of an EWF net can be checked on the reachability graph of the workflow net (Proposition 4.1). Hence, checking soundness for EWF nets can be done using standard reachability algorithms, which are already implemented in tools like Woflan [138]. In the remainder of the chapter we assume that a soundness check has been successful and consider only sound EWF nets.

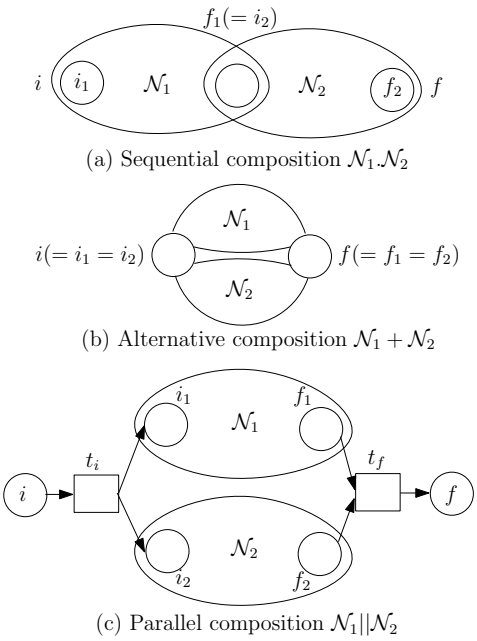
Weak soundness

Weak soundness [98] is a weaker version of soundness, by which an EWF net has the option to complete and all transitions are quasi-live.

Definition 4.4. [Weak Soundness of Exception Workflow Nets] *An exception workflow net $\mathcal{N}$ is* weakly sound *iff there exists $\sigma \in T_n^*$ so that $[i] \overset{\sigma}{\Longrightarrow} [(f, \mathfrak{b})]$ and $\mathcal{N}$ is quasi-live.*

Relaxed soundness

Relaxed soundness [43] is a weaker version of soundness, by which all transitions are included in some proper terminating run. For WF nets it is possible to transform a relaxed sound net into a sound net [43]. Relaxed soundness is an important concept since it is composable. In the context of adaptive nets relaxed soundness can be used for compositional verification of soundness.



(a) Sequential composition $\mathcal{N}_1.\mathcal{N}_2$

(b) Alternative composition $\mathcal{N}_1 + \mathcal{N}_2$

(c) Parallel composition $\mathcal{N}_1||\mathcal{N}_2$

Fig. 10: Operations on exception workflow nets

Definition 4.5. [Relaxed Soundness of Exception Workflow Nets] *An exception workflow net $\mathcal{N}$ is* relaxed sound *iff for all $t \in T_n$, there exist $M, M' \in \mathcal{R}(\mathcal{N}, [i])$, $\sigma \in T_n^*$, $\sigma' \in T_n^*$ so that $[i] \stackrel{\sigma}{\Longrightarrow} M \stackrel{t}{\Longrightarrow} M' \stackrel{\sigma'}{\Longrightarrow} [f]$, and $\mathcal{N}$ is t-quasi-live for all $t \in T_e$.*

Operations on exception workflow nets

Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two exception workflow nets nets. We use the following operations on exception workflow nets nets: parallel composition $\mathcal{N}_1||\mathcal{N}_2$, alternative composition $\mathcal{N}_1 + \mathcal{N}_2$ and sequential composition $\mathcal{N}_1.\mathcal{N}_2$.

Definition 4.6. [sequential/parallel/alternative composition] *Let $\mathcal{N}_1 = \langle P^1, T_n^1 \cup T_e^1, A_1, l^1 \rangle$, $\mathcal{N}_2 = \langle P^2, T_n^2 \cup T_e^2, A_2, l^2 \rangle$ be two EWF nets. Then $\mathcal{N}_1.\mathcal{N}_2$, $\mathcal{N}_1 + \mathcal{N}_2$ and $\mathcal{N}_1||\mathcal{N}_2$ are EWF nets defined as follows:*

sequential composition

$\mathcal{N}_1.\mathcal{N}_2 \stackrel{\text{def}}{=} \langle P^1 \cup P^2 \cup \{i, f\} \setminus \{i_1, i_2, f_2\}, (T_n^1 \cup T_n^2) \cup (T_e^1 \cup T_e^2), A, l \rangle$, *where*

1. $A = A_1' \cup A_2' \cup A''$ *with*
   - $A_1' = A_1 \setminus \{(i_1, t) | t \in (i_1^\bullet)_{\mathcal{N}_1}\}$,
   - $A_2' = A_2 \setminus \{(i_2, t) | t \in (i_2^\bullet)_{\mathcal{N}_2}\} \setminus \{(t, f_2) | t \in ({}^\bullet f_2)_{\mathcal{N}_2}\}$,

- $A'' = \{(i,t) | t \in (i_1^\bullet)_{\mathcal{N}_1}\} \cup \{(t,f) | t \in ({}^\bullet f_2)_{\mathcal{N}_2}\} \cup \{(f_1,t) | t \in (i_2^\bullet)_{\mathcal{N}_2}\}$ *and*

2. $l = l^1 \cup l^2$.

alternative composition

$\mathcal{N}_1 + \mathcal{N}_2 \stackrel{\text{def}}{=} \langle P, (T_n^1 \cup T_n^2) \cup (T_e^1 \cup T_e^2), A, l \rangle$, *where*

1. $P = P^1 \cup P^2 \cup \{i,f\} \setminus \{i_1, f_1, i_2, f_2\}$;
2. $A = A_1' \cup A_2' \cup A''$ *with*
   - $A_1' = A_1 \setminus (\{(i_1,t) | t \in (i_1^\bullet)_{\mathcal{N}_1}\}) \cup \{(t,f_1) | t \in ({}^\bullet f_1)_{\mathcal{N}_1}\})$,
   - $A_2' = A_2 \setminus (\{(i_2,t) | t \in (i_2^\bullet)_{\mathcal{N}_2}\} \cup \{(t,f_2) | t \in {}^\bullet f_2)_{\mathcal{N}_2}\})$,
   - $A'' = \{(i,t) | t \in (i_1^\bullet)_{\mathcal{N}_1} \cup (i_2^\bullet)_{\mathcal{N}_2}\} \cup \{(t,f) | t \in ({}^\bullet f_1)_{\mathcal{N}_1} \cup ({}^\bullet f_2)_{\mathcal{N}_2}\}$ *and*
3. $l = l^1 \cup l^2$.

parallel composition

$\mathcal{N}_1 || \mathcal{N}_2 \stackrel{\text{def}}{=} \langle P^1 \cup P^2 \cup \{i,f\}, (T_n^1 \cup T_n^2 \cup \{t_i, t_f\}) \cup (T_e^1 \cup T_e^2), A, l \rangle$, *where*

1. $A = A_1 \cup A_2 \cup A'$, $A' = \{(i,t_i), (t_i,i_1), (t_i,i_2), (f_1,t_f), (f_2,t_f), (t_f,f)\}$,
2. $l = l^1 \cup l^2 \cup (\{t_i\} \to \{\tau\}) \cup (\{t_f\} \to \{\tau\})$.

These operations are depicted in Figure 10 and resemble standard operations from process algebra [23][3].

Lemma 4.2. *Let N_1 and N_2 be two sound EWF nets and $\phi \in \{.,||,+\}$. Then $N_1 \phi N_2$ is (weakly/relaxed) sound as well.*

Proof. Let $M = M_1 + M_2$ be a marking of $N_1 \phi N_2$ such that $[i] \stackrel{\sigma}{\Longrightarrow} M$, with $\sigma \in (T_n^1 \cup T_n^2)^*$, $M_1 = \sum_{p \in P_1} M(p)$ and $M_2 = \sum_{p \in P_2} M(p)$.

$N_1.N_2$ Since having $M_1 \neq \emptyset$ and $M_2 \neq \emptyset$ would imply unsoundness of N_1, we have two cases 1) $M_1 \neq \emptyset$ and $M_0 = \emptyset$; 2) $M_1 = \emptyset$ and $M_2 \neq \emptyset$. In the first case, by soundness of N_1, we obtain $M_1 \stackrel{\sigma'}{\Longrightarrow} [f_1] = [i_2] \stackrel{\sigma''}{\Longrightarrow} [f]$, where σ' and $\sigma'' \in (T_n^2)^*$. The second case is analogous. Quasi-liveness of $(N_1, [i_1])$ implies that for all $t \in T_n^1 \cup T_e^1$, $i \stackrel{*}{\Longrightarrow} M \stackrel{t}{\longrightarrow}$, hence they are quasi-live in $(N_1.N_2, [i])$. Similarly, by quasi-liveness of $(N_2, [i_2])$, we have that for all $t \in T_n^2 \cup T_e^2$, $[i_2] \stackrel{*}{\Longrightarrow} M \stackrel{t}{\longrightarrow}$ in N_2, hence $[i_1] \stackrel{*}{\Longrightarrow} [i_2] \stackrel{*}{\Longrightarrow} M \stackrel{t}{\longrightarrow}$ in $(N_1.N_2, [i])$. Thus $(N_1.N_2, [i])$ is quasi-live.

$N_1 + N_2$ We have two cases: either $M = M_1$ or $M = M_2$. For the first case, since N_1 is sound $M \stackrel{*}{\Longrightarrow} [f_1] = [f]$ in N_1, for $\sigma' \in (T_n^1)^*$, hence $M \stackrel{*}{\Longrightarrow} [f_1] = [f]$ in $N_1 + N_2$. The second case can be treated analogously. Since for all $t \in T_n^1 \cup T_e^1$, there exists an M such that $[i] \stackrel{*}{\Longrightarrow} M \stackrel{t}{\Longrightarrow}$ in N_1, we have that $[i] \stackrel{*}{\Longrightarrow} M \stackrel{t}{\Longrightarrow}$ in $N_1 + N_2$. Quasi-liveness of $t \in T_n^2 \cup T_e^2$ in $(N_1 + N_2, [i])$ can be proven analogously.

[3] Parallel composition differs from the corresponding process algebraic operator as its execution starts with an extra transition firing.

$N_1||N_2$ In this case $M_1 \neq \varnothing$ and $M_2 \neq \varnothing$, and we can write $[i] \stackrel{t_i}{\Longrightarrow} [i_1]+[i_2] \stackrel{\sigma}{\Longrightarrow} M_1 + [i_2] \stackrel{\sigma'}{\Longrightarrow} M_1 + M_2$, where $\sigma \in (T_n^1)^*$ and $\sigma' \in (T_n^2)^*$. By soundness of N_1 and N_2, $M_1 \stackrel{\sigma_1}{\Longrightarrow} [f_1]$ ($\sigma_1 \in (T_n^1)^*$) and $M_2 \stackrel{\sigma_2}{\Longrightarrow} [f_2]$ ($\sigma_2 \in (T_n^2)^*$), hence $M_1 + M_2 \stackrel{\sigma_1\sigma_2}{\Longrightarrow} [f_1] + [f_2]$ in $N_1||N_2$. Therefore, $M \stackrel{\sigma_1\sigma_2 t_f}{\Longrightarrow} [f]$ in $N_1||N_2$. Quasi-liveness of $t \in \{t_i\} \cup T_n^1 \cup T_e^1 \cup T_n^1 \cup T_e^1$ in $(N_1||N_2, [i])$ follows from $[i] \stackrel{t_i}{\Longrightarrow} [i_1] + [i_2]$ and the quasi-liveness of $t \in T_n^1 \cup T_e^1$ in $(N_1, [i_1])$ and $t' \in T_n^2 \cup T_e^2$ in $(N_2, [i_2])$, respectively.

Relaxed/Weak soundness of $N_1 \phi N_2$ can be proven in the same way.

Operations on marked EWF nets

Definition 4.7. [INITIALIZATION]
The initialization *of an exception workflow net $\mathcal{N} = \langle P, T_n \cup T_e, A, l \rangle$, denoted by $init(\mathcal{N})$, is the marked exception workflow net $(\mathcal{N}, [i])$.*

We extend two operations for the case of *marked* exception workflow nets: the sequential composition of a marked EWF $(\mathcal{N}_1, M)$ with an EWF $\mathcal{N}_2$, i.e. the running process $(\mathcal{N}_1, M)$ is extended with the functionality of $\mathcal{N}_2$ and the parallel composition of two marked exception workflow nets $(\mathcal{N}_1, M_1)$ and $(\mathcal{N}_2, M_2)$, i.e. two running processes are set to synchronize upon termination.

Definition 4.8. [OPERATIONS ON MARKED EWFS]
Let $\mathcal{N}_1 = \langle P^1, T_n^1 \cup T_e^1, A_1, l^1 \rangle$, $\mathcal{N}_2 = \langle P^2, T_n^2 \cup T_e^2, A_2, l^2 \rangle$ be two EWF nets.

1. *The sequential composition of a marked EWF net $(\mathcal{N}_1, M)$ with an EWF net $\mathcal{N}_2$ is a marked EWF net $(\mathcal{N}_1, M).\mathcal{N}_2 \stackrel{\text{def}}{=} (\mathcal{N}_1.\mathcal{N}_2, M)$.*
2. *The parallel composition of two marked EWF nets $(\mathcal{N}_1, M_1)$ and $(\mathcal{N}_2, M_2)$ is a marked EWF net $(\mathcal{N}_1, M_1)||(\mathcal{N}_2, M_2) \stackrel{\text{def}}{=} (\mathcal{N}_1||\mathcal{N}_2, M_1 + M_2)$.*

We also say in this case that the exception workflow net $(\mathcal{N}, M_0)$ has the *initial marking* if $M_0 = [i]$, where i is the initial place of $\mathcal{N}$. In case $(\mathcal{N}, M)$ is the parallel composition of two exception workflow nets $(\mathcal{N}_1, M_1)$ and $(\mathcal{N}_2, M_2)$, we consider $M_0 = [i_1] + [i_2]$ as *initial marking*, where i_1 and i_2 are the initial places for $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively.

Lemma 4.3. *Let (N_1, M_0^1) and (N_2, M_0^2) be two EWF nets with their initial marking, let M_0 be the initial marking of the marked EWF net obtained by parallel composition $(N_1, M_0^1)||(N_1, M_0^2)$, $T = T_n \cup T_e$ its set of transitions and $i^\bullet = [t_i]$ in $N_1||N_2$.*

1. *If N_1 and N_2 are sound, then $(N_1||N_1, M_0^1 + M_0^2)$ satisfies the following properties:*
   (a) *For all M such that $M_0 \stackrel{\sigma}{\Longrightarrow} M$, for some transition sequence σ that does not contain any exception transition, i.e. $\sigma \in T_n \setminus \{t_i\}^*$, there exists $\sigma' \in T_n^*$ such that $M \stackrel{\sigma'}{\Longrightarrow} [f]$.*

(b) $(N_1, M_0^1)\|(N_2, M_0^2)$ *is* $T \setminus \{t_i\}$*-quasi-live.*

2. *If* N_1 *and* N_2 *are relaxed sound, then there exists* $\sigma \in T_n^*$ *such that* $M_0 \stackrel{\sigma}{\Longrightarrow} [f]$ *and* $(N_1, M_0^1)\|(N_2, M_0^2)$ *is* $T \setminus \{t_i\}$*-quasi-live.*
3. *If* N_1 *and* N_2 *are weakly sound, then there exists* $\sigma \in T_n^*$ *such that* $M_0 \stackrel{\sigma}{\Longrightarrow} [f]$ *and* $(N_1, M_0^1)\|(N_2, M_0^2)$ *is* $T \setminus \{t_i\}$*-quasi-live.*

Proof. Follows from Lemma 4.2. □

Lemma 4.3 can be extended for the parallel composition of marked sound nets to guarantee proper termination of markings reachable by non-exceptional transitions from the sum of initial markings when no exception transitions are fired. Similarly, the sequential composition of a marked sound EWF net with a sound EWF net always guarantees proper termination (by non-exceptional transition sequences) of markings reachable from the initial marking of the result net by non-exceptional transitions. Other possible operations either do not guarantee this property (e.g. alternative composition of marked EWF nets) or are not feasable (e.g. an EWF net which is composed sequentially with a marked EWF net does not add to the behavior), therefore we exclude them out of consideration.

4.3.2 Adaptive Workflow Nets

Intuitively, an adaptive workflow net is an exception workflow net where tokens can be exception workflow nets themselves and firings of a higher-level net can depend on firings in the token nets. To introduce *adaptive workflow nets*, we extend first the notion of exception workflow nets to incorporate net expressions. We call this type of nets *extended workflow nets*.

Let $Var = \{v, \dots\}$ be a finite set of *variable* names, *Con* a finite set of *constant* names, $\mathfrak{b} \notin Con$ is a constant denoting the black token and Σ^e a finite set of exception labels. We introduce the following language of expressions:

Definition 4.9. [Expression]
A net expression e *and a* token expression te *are inductively defined as:* $e := c \mid e + e \mid e\|e \mid e.e$, $ce := v \mid ce\|ce \mid ce.e \mid init(e)$, *and* $te := \mathfrak{b} \mid ce$, *where* $v \in Var$, $c \in Con$.

The sets of all net expressions and token expressions are denoted by *Expr* and *TExpr*, respectively. The expressions in *Expr* will be interpreted as nested workflow nets while the expressions in *TExpr* denote either black tokens ($\mathfrak{b}$) or marked nested workflow nets.

Firings of adaptive nets can be synchronized with the firings in net tokens, which is modeled by guards of transitions that are expressed in the guard language $\mathcal{GL}$.

Definition 4.10. [Guard]
A guard g *is defined as* $g := \top \mid e(v)$, *where* $v \in Var$, *and labels* $e \in \Sigma^e$. *A guard* $e(v)$ *is called an* exception guard. *The set of all guards is denoted by* $\mathcal{GL}$.

Intuitively, the guard $\top$ of a transition t means that the firing of t does not depend on the internal states of the net tokens, $e(v)$ means that the firing of t is conditioned by the firing of an exception transition with label e in the token net v.

We overload the notation for the set of variables (Var) and constants (Con) and denote the set of variables and constants appearing in an expression $e \in TExpr \cup \mathcal{GL}$ by $Var(e)$ and $Con(e)$, respectively.

We now define extended workflow nets as EWF nets.

Definition 4.11. [Extended workflow net]
An extended workflow net $\mathcal{N}$ *is a tuple* $\langle P, T, A, \mathcal{E}, g, l\rangle$*, where* $\langle P, T, A, l\rangle$ *is an EWF net called* system net *and the extensions* $\mathcal{E}, g$ *are defined as follows:*

- $\mathcal{E}: A \to TExpr$ *are* arc expressions *such that*
  1. *All input arcs of transitions are mapped either to the black token or to variables, i.e. for every* $(p,t) \in A \cap (P \times T)$, $\mathcal{E}(p,t) \in Var \cup \{\mathfrak{b}\}$*;*
  2. *Every two variables on two different input arcs of a transition are distinct, i.e.* $Var(\mathcal{E}(p,t)) \cap Var(\mathcal{E}(p',t)) = \varnothing$ *for all* $(p,t), (p',t) \in A \cap (P \times T)$ *with* $p \neq p'$*;*
  3. *Every variable on the outgoing arc of a transition occurs also in the expression of some incoming arc of this transition, i.e. for all* $(t,p) \in A \cap (T \times P)$, $v \in Var(\mathcal{E}(t,p))$ *implies* $v \in Var(\mathcal{E}(p',t))$ *for some* $(p',t) \in A \cap (P \times T)$*;*
  4. *All outgoing arcs of the initial place and incoming arcs of the final place are mapped to the black token, i.e. for all* $t \in i^\bullet$, $\mathcal{E}(i,t) = \mathfrak{b}$ *and for all* $t \in {}^\bullet f$, $\mathcal{E}(t,f) = \mathfrak{b}$.
- g *is a function that maps transitions from* T *to expressions from* $\mathcal{GL}$ *such that the variable of a guard* $g(t)$ $(t \in T)$ *appears in the expression of some incoming arc of* t *and does not appear in any outgoing arc of* t*, i.e.* $Var(g(t)) \subseteq \bigcup_{p \in {}^\bullet t} Var(\mathcal{E}(p,t))$ *and* $Var(g(t)) \cap \bigcup_{p \in t^\bullet} Var(\mathcal{E}(t,p)) = \varnothing$.

Let $\mathfrak{W}$ be the set of all extended workflow nets. We presuppose the existence of a library $\mathcal{L}$ which maps constants to extended workflow nets. Since in our application domain there is no need for recursive calls, e.g. a net has a constant in one of its arc expression mapped to itself, we define inductively the family of sets of extended workflow nets $(\mathfrak{N}_j)_{j \geq 0}$, and marked extended workflow nets $(\mathfrak{M}_j)_{j \geq 0}$, respectively:

1. We define $\mathfrak{N}_0 \stackrel{\text{def}}{=} \varnothing$ and $\mathfrak{N}_1 \stackrel{\text{def}}{=} \{\langle P, T, A, \mathcal{E}, g, l\rangle \in \mathfrak{W} \mid \mathcal{E}: A \to \{\mathfrak{b}\} \wedge g: T \to \{\top\}\}$, which is in fact the set of all exception workflow nets trivially extended with the black-colored tokens and $\top$ transition guards.
   $\langle P, T, A, \mathcal{E}, g, l\rangle \in \mathfrak{N}_{k+1}$, for $k \geq 1$, iff for all $a \in A$ and $c \in Con(\mathcal{E}(a))$, $\mathcal{L}(c) \in \mathfrak{N}_k$.
2. We define $\mathfrak{M}_0 \stackrel{\text{def}}{=} \varnothing$. A marking M of $\mathcal{N} = \langle P, T, A, \mathcal{E}, g, l\rangle \in \mathfrak{N}_k$, $k \geq 1$ is a bag over $P \times (\mathfrak{M}_{k-1} \cup \{\mathfrak{b}\})$.
   $\mathfrak{M}_{k+1} \stackrel{\text{def}}{=} \{(N, M) | N = \langle P, T, A, \mathcal{E}, g, l\rangle \in \mathfrak{N}_{k+1} \wedge M \in \mathbb{N}^{P \times (\mathfrak{M}_k \cup \{\mathfrak{b}\})}\}$ is the set of marked extended workflow nets with tokens from $\mathfrak{M}_k \cup \{\mathfrak{b}\}$, for $k \geq 0$.

Note that $\mathfrak{N}_j \subseteq \mathfrak{N}_{j+1}$ and $\mathfrak{M}_j \subseteq \mathfrak{M}_{j+1}$, for all $j \geq 0$. Let $j \geq 0$. The sets $\mathfrak{N}_{j+1}$ and $\mathfrak{M}_{j+1}$ represent the set of extended workflow net and marked extended workflow nets of level up to $j+1$, i.e. which have constants in arc expressions mapped to extended workflow nets of level up to j and black tokens, respectively.

We overload the notation and write $M(p)$ for $\sum_{(p,\nu)\in M} M(p,\nu)[\nu]$, i.e. the bag of black/net tokens present in place p at the marking M.

Definition 4.12. [Adaptive workflow nets]
An extended workflow net $\mathcal{N} \in \mathfrak{N}_k \setminus \mathfrak{N}_{k-1}$ ($k \geq 1$) is called an adaptive workflow net of level k. *A* marked adaptive workflow net of level k, *for some $k \geq 1$, is a couple $(\mathcal{N}, M_{\mathcal{N}})$, where $\mathcal{N}$ is an adaptive workflow net of level k and $M_{\mathcal{N}}$ is a marking of $\mathcal{N}$.*

In the remainder, we will also call adaptive workflow nets shortly *adaptive nets*.

Note that this class is a subclass of the class of adaptive nets defined in [62] which allow for other tokens than black tokens and net tokens and have guards are not parametrized by data values. The class of nested nets of Lomazova [94] allow recursiveness, i.e. a nested net can call an instance of itself, which is not a feature of adaptive nets in [62] and a more complicated synchronization mechanism but no operations on nets for nested nets are less expressive with respect to operations on colored tokens (which make the class in [62] Turing complete).

We extend the operations on EWF nets from Definitions 4.6, 4.7, 4.8 to operations on adaptive nets in a natural way.

Firings of an adaptive net take into account the current marking and an assignment of net tokens to the variables in net expression on arcs w.r.t. the current marking, called *binding*. Let $\mathcal{N} = \langle P, T, A, \mathcal{E}, g, l\rangle \in \mathfrak{N}_k$, for $k \geq 1$. A binding is a mapping $b\colon Var \to \mathfrak{M}_{k-1}$. We denote the set of all bindings by $\mathcal{B}$. Let $e \in Expr \cup TExpr$. We lift bindings $b \in \mathcal{B}$ to expressions from $\{\mathfrak{b}\} \cup Expr \cup TExpr$ as follows:

$$b(e) = \begin{cases} \mathcal{L}(e) & \text{if } e = c \in Con, \\ \mathfrak{b} & \text{if } e = \mathfrak{b}, \\ init(b(e')) & \text{if } e = init(e'), \text{ where } e' \in Expr, \\ b(e_1)\rho\, b(e_2) & \text{if } e = e_1\ \rho\ e_2, \text{ where } e_1, e_2 \in Expr \cup TExpr, \rho \in \{.,||,+\}. \end{cases}$$

For any binding $b \in \mathcal{B}$ we have $b(e) \in \mathfrak{N}_{k-1}$ if $e \in Expr$ and $b(e) \in \mathfrak{M}_{k-1}$ if $e \in TExpr$.

Next we inductively define the firing relation $\longrightarrow \subseteq \mathbb{N}^{P\times(\mathfrak{M}_{k-1}\cup\{\mathfrak{b}\})} \times (\{\tau\} \cup (T \times \mathcal{B})) \times \mathbb{N}^{P\times(\mathfrak{M}_{k-1}\cup\{\mathfrak{b}\})}$, where $\tau \notin \bigcup_{t\in T} l(t)$. We write $M \xrightarrow{(t,b)} M'$ iff $(M,(t,b),M') \in \longrightarrow$ and $M \xrightarrow{\tau} M'$ iff $(M,\tau,M') \in \longrightarrow$.

1. For $(\mathcal{N}, M) \in \mathfrak{M}_1$, $\xrightarrow{(t,b)}$ coincides with the PN firing relation $\xrightarrow{t}$ for any $b \in \mathcal{B}$.
2. Let $\longrightarrow$ be defined for all marked adaptive nets of level $j-1$, where $j \geq 2$. Then for a marked adaptive net $(\mathcal{N}, M)$ of level j, the firing is either (a) a net firing or (b) a net token transformation:
   (a) Let $t \in T$, $b \in \mathcal{B}$, $b(\mathcal{E}(p,t)) \in M(p)$ for all $p \in {}^\bullet t$ and one of the following conditions holds:
   - $g(t) = \top$;
   - $g(t) = \mathit{final}(v)$, $\mathcal{E}(p',t) = v$ for some $p' \in {}^\bullet t$ and $b(v) = (\mathcal{N}', [(f, \mathfrak{b})])$ for some $\mathcal{N}' \in \mathfrak{N}_{j-1}$;
   - $g(t) = e(v)$, $\mathcal{E}(p',t) = v$ for some $p' \in {}^\bullet t$, $b(v) = (N', M_{\mathcal{N}'})$, for some $(\mathcal{N}', M_{\mathcal{N}'}) \in \mathfrak{M}_{j-1}$, and $M' \xrightarrow{(t',b)}$ for some exception transition t' with $l(t') = e$ and binding b'.

   Then $M \xrightarrow{(t,b)} M'$, where

$$M' = M - \sum_{p \in {}^\bullet t} [(p, b(\mathcal{E}(p,t)))] + \sum_{p \in t^\bullet} [(p, b(\mathcal{E}(t,p)))].$$

   (b) Let $M = M' + [(p, (\mathcal{N}', M_{\mathcal{N}'}))]$ for some place p in $\mathcal{N}$, and $M_{\mathcal{N}'} \xrightarrow{x} M'_{\mathcal{N}'}$ in $\mathcal{N}'$, where x is either τ or some (t', b'). Then $M \xrightarrow{\tau} M' + [(p, (\mathcal{N}', M'_{\mathcal{N}'}))]$.

We will use the following shorthand notation: We write $M \xrightarrow{*} M'$ iff M' is reachable from M by a (possibly empty) sequence of firings, $M \xrightarrow{t}$ if there exists a marking M' such that $M \xrightarrow{t} M'$, $M \xrightarrow{\tau^*} M'$ when $M \xrightarrow{\tau^n}$ for some $n \in \mathbb{N}$, and $M \overset{t}{\Longrightarrow} M'$ if there exists a binding $b \in \mathcal{B}$ such that $M \xrightarrow{\tau^*} M_1 \xrightarrow{(t,b)} M_2 \xrightarrow{\tau^*} M'$. If $\sigma = t_1 \dots t_k \in T^+$, for $k \geq 1$, we write $M \overset{\sigma}{\Longrightarrow} M'$ iff $M \overset{t_1}{\Longrightarrow} M_1 \overset{t_2}{\Longrightarrow} \dots \overset{t_k}{\Longrightarrow} M'$, $M \overset{\sigma}{\Longrightarrow}$ if there exists M' so that $M \overset{\sigma}{\Longrightarrow} M'$.

The notions of quasi-liveness, set of reachable markings of a marked adaptive net $(\mathcal{N}, [(i, \mathfrak{b})])$ and the set of markings of $\mathcal{N}$ that can reach $[(f, \mathfrak{b})]$, denoted by $\mathcal{R}(\mathcal{N}, [(i, \mathfrak{b})])$ and $\mathcal{S}(\mathcal{N}, [(f, \mathfrak{b})])$, resemble Definitions 2.23 and 2.25, respectively.

Figure 8 shows an adaptive workflow net of level 2, where the constants refer to the following EWF nets modeling different test and treatment procedures: $\mathit{STests}\langle \mathit{partial\ response}, \mathit{relapse} \rangle_e$ with exception labels *partial response* and *relapse*; *Cisplatin*; *Etoposide*; $\mathit{RadControl}\langle \mathit{radiation\ scarring} \rangle_e$ with exception label *radiation scarring*; *Radiotherapy* and *Prophylactic*. Transition t_4 with guard g_3 can fire only when the exception transition labeled by *radiation scarring* is enabled in the net token on place q and there is a net token in place p, and this firing results in two new net tokens: $s_3 || \mathit{init}(\mathit{Prophylactic})$, i.e., actual treatment being executed in parallel with a prophylactic treatment, and a new $\mathit{RadControl}\langle \mathit{radiation\ scarring} \rangle_e$ net token.

The main protocol (Figure 9) is an adaptive net of level 3.

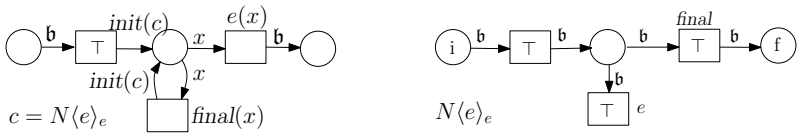


Fig. 11: Unsound adaptive net

4.3.3 Correctness Properties of Adaptive Workflow Nets

Soundness

Soundness is an important property of adaptive workflow nets stating that at any moment of system run there is a chance to terminate properly. With proper termination we mean the possibility to reach the final marking without encountering any exceptions or synchronizing on exceptions. This is a desirable property in real systems since there should be a possibility to terminate properly also when no exception occurs.

Definition 4.13. [Soundness of adaptive nets]
An adaptive net $\mathcal{N}$ is called sound *iff*

1. *For all M such that $[(i,\mathfrak{b})] \overset{\sigma}{\Longrightarrow} M$, for some transition sequence σ that does not contain any exception transition, i.e. $\sigma \in T_n^*$, there exists $\sigma' \in T_n^*$ such that $M \overset{\sigma'}{\Longrightarrow} [(f,\mathfrak{b})]$, and for all t from σ', $g(t) \in \{final(v), \top\}$.*
2. *$\mathcal{N}$ is quasi-live.*

Note that we require σ to be a sequence from T_n^* since any firing of a transition from T_e terminates the execution. The requirement $g(t) \in \{final(v), \top\}$ expresses the absence of transitions synchronizing o exceptions in σ'.

Our notion of soundness requires that the process is able to terminate properly also when the current state is reached after synchronizing on exceptions that happen at a lower level (in some net token). The adaptive net in Figure 11 creates a new net token — a protocol $N\langle e\rangle_e$ that can trigger the exception labeled by e. However, the adaptive net can only terminate properly when this exception is triggered (the transition with the guard $e(x)$ is fired), which violates soundness. This does not capture the notion of soundness which means proper termination of non-exceptional behavior.

We define weak soundness and relaxed soundness of adaptive nets.

Definition 4.14. [Weak Soundness of Adaptive Workflow Nets]
An adaptive workflow net $\mathcal{N}$ is weakly sound *iff there exists $\sigma \in T_n^*$ so that for all t from σ', $g(t) \in \{final(v), \top\}$, $[(i,\mathfrak{b})] \overset{\sigma}{\Longrightarrow} [(f,\mathfrak{b})]$ and $\mathcal{N}$ is quasi-live.*

Definition 4.15. [Relaxed Soundness of Adaptive Workflow Nets]
An adaptive workflow net $\mathcal{N}$ is relaxed sound *iff for all $t \in T_n$, there exist $M, M' \in \mathcal{R}(\mathcal{N}, [(i,\mathfrak{b})])$, $\sigma \in T_n^*$, $\sigma' \in T_n^*$ so that for all t from σ', $g(t) \in$*

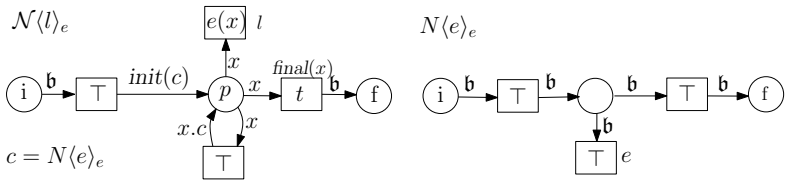


Fig. 12: Sound adaptive net, with infinite set of reachable markings

$\{final(v), \top\}$, *with* $[(i, \mathfrak{b})] \overset{\sigma}{\Longrightarrow} M \overset{t}{\Longrightarrow} M' \overset{\sigma'}{\Longrightarrow} [(f, \mathfrak{b})]$, *and* $\mathcal{N}$ *is* t*-quasi-live for all* $t \in T_e$.

Note that we work with a library $\mathcal{L}$ of sound adaptive nets, and the net tokens remain sound also when we construct new nets from net tokens. Moreover relaxed and weak soundness is preserved by the compositions that we have described.

Lemma 4.4. *Let* $\mathcal{L}$ *be a library of (weakly/relaxed) sound adaptive workflow nets. Any net expression* $e \in Expr$ *evaluated over nets from* $\mathcal{L}$ *results in a (relaxed, weakly) sound adaptive workflow net.*

Proof. We prove the statement by structural induction on the expression e. We focus on the case where $e = e_1 \phi e_2$, for some b such that $b(e_1) = N_1$ and $b(e_2) = N_2$ are two sound adaptive nets and $\phi \in \{., ||, +\}$. For each case the proof is analogous to Lemma 4.2. □

Lemma 4.5. *Let* (N_1, M_0^1) *and* (N_2, M_0^2) *be two adaptive nets with their initial marking, let* M_0 *be the initial marking of the marked adaptive net obtained by parallel composition* $(N_1, M_0^1)||(N_1, M_0^2)$, $T = T_n \cup T_e$ *its set of transitions and* $i^\bullet = [t_i]$ *in* $N_1||N_2$.

1. *If* N_1 *and* N_2 *are sound, then* $(N_1||N_1, M_0^1 + M_0^2)$ *satisfies the following properties:*
   (a) *For all* M *such that* $M_0 \overset{\sigma}{\Longrightarrow} M$, *for some transition sequence* σ *that does not contain any exception transition, i.e.* $\sigma \in T_n \setminus \{t_i\}^*$, *there exists* $\sigma' \in T_n^*$ *such that* $M \overset{\sigma'}{\Longrightarrow} [(f, \mathfrak{b})]$, *and for all* t *from* σ', $g(t) \in \{final(v), \top\}$.
   (b) $(N_1, M_0^1)||(N_2, M_0^2)$ *is* $T \setminus \{t_i\}$*-quasi-live.*
2. *If* N_1 *and* N_2 *are relaxed sound, then there exists* $\sigma \in T_n^*$ *such that for all* t *from* σ', $g(t) \in \{final(v), \top\}$, $M_0 \overset{\sigma}{\Longrightarrow} [(f, \mathfrak{b})]$ *and* $(N_1, M_0^1)||(N_2, M_0^2)$ *is* $T \setminus \{t_i\}$*-quasi-live.*
3. *If* N_1 *and* N_2 *are weakly sound, then there exists* $\sigma \in T_n^*$ *such that for all* t *from* σ', $g(t) \in \{final(v), \top\}$, $M_0 \overset{\sigma}{\Longrightarrow} [(f, \mathfrak{b})]$ *and* $(N_1, M_0^1)||(N_2, M_0^2)$ *is* $T \setminus \{t_i\}$*-quasi-live.*

Proof. Follows from Lemma 4.4. □

Boundedness

Figure 12 shows an adaptive net $(\mathcal{N}\langle l\rangle_e)$ for which $\mathcal{R}(\mathcal{N}\langle l\rangle_e, [(i,\mathfrak{b})])$ is infinite, since

$$\{[(p,(N\langle e\rangle_e^{(n)}, [(i,\mathfrak{b})]))] | n \geq 1\} \subseteq \mathcal{R}(\mathcal{N}\langle l\rangle_e, [(i,\mathfrak{b})]),$$

where $N\langle e\rangle_e^{(1)} = N\langle e\rangle_e$ and $N\langle e\rangle_e^{(j+1)} = N\langle e\rangle_e^{(j)}.N\langle e\rangle_e$. However, the net is sound since from each $M \in \mathcal{R}(\mathcal{N}\langle l\rangle_e, [(i,\mathfrak{b})])$, $M \stackrel{*}{\Longrightarrow} [(f,\mathfrak{b})]$. Hence, unlike for exception workflow nets, the reachability set of a sound adaptive net can be infinite.

Note that the net in Figure 12 can reach the final marking from an infinite set of markings, i.e. $[(p,(N\langle e\rangle_e^{(n)}, [(f,\mathfrak{b})]))] \stackrel{t}{\Longrightarrow} [(f,\mathfrak{b})]$, for all $n \in \mathbb{N}^*$ (since each $N\langle e\rangle_e^{(n)}$ is sound, for all $n \in \mathbb{N}^*$). Hence our class of adaptive nets is more expressive than classical Petri nets.

Definition 4.16 (Boundedness of adaptive nets).
An adaptive net $\mathcal{N}$ is called bounded *iff $\mathcal{R}(\mathcal{N}, [(i,\mathfrak{b})])$ is finite, and* unbounded *otherwise.*

To capture the variety of tokens that can be present on a place, we introduce the notion of place support.

Definition 4.17 (Support of a place).
Let $\mathcal{N}$ be an adaptive net of level $k \geq 1$. Then the place support *is the function $Supp: P \to 2^{\mathfrak{M}_{k-1} \cup \{\mathfrak{b}\}}$ defined as*

$$Supp(p) \stackrel{\text{def}}{=} \{\nu | \exists M \in \mathcal{R}(\mathcal{N}, [(i,\mathfrak{b})]) \wedge (p,\nu) \in M\}.$$

To characterize the size of adaptive nets we introduce a mapping for nets to natural numbers, i.e. $size(N): \mathfrak{N} \to \mathbb{N}$ given by $size(N) = |P| \times |T|$ for $N = \langle P, T, A, \mathcal{E}, g, l\rangle \in \mathfrak{N}$.

Lemma 4.6. *Let $\mathcal{N} = \langle P, T, A, \mathcal{E}, g, l\rangle$ be an adaptive net. Then $\mathcal{N}$ is unbounded iff at least one of the conditions hold:*

1. *There exist two markings $M, M' \in \mathcal{R}(\mathcal{N}, [(i,\mathfrak{b})])$ of $\mathcal{N}$ and $\sigma \in T^*$ such that $M \stackrel{\sigma}{\Longrightarrow} M'$ and $M < M'$.*
2. *The support $Supp(p)$ of some place $p \in P$ of $\mathcal{N}$ is an infinite set.*

Proof. ($\Rightarrow$) We proceed by induction on the level of the adaptive net.

k=1 If $\mathcal{N}$ is unbounded then there are two markings M, M' reachable from the initial marking with $M \stackrel{\sigma}{\Longrightarrow} M'$ and $M < M'$ [118].

k>1 Suppose that $\mathcal{N}$ is unbounded and $Supp(p)$ is finite for all $p \in P$. Since $\mathcal{N}$ is unbounded, $\mathcal{R}(\mathcal{N}, [(i,\mathfrak{b})])$ is infinite. Due to the finiteness of the number of places and the variety of tokens on them, we can apply Dickson's Lemma (Lemma 2.1). Therefore there exists an infinite sequence of markings $M_1 < M_2 < \cdots$ such that $[(i,\mathfrak{b})] \stackrel{*}{\Longrightarrow} M_1 \stackrel{*}{\Longrightarrow} M_2 \stackrel{*}{\Longrightarrow} \cdots$ and the first condition holds.

($\Leftarrow$) We have two cases:

1. Since $M < M'$, there is a $\Delta \in \mathfrak{M}_k$, $\Delta \neq \varnothing$, such that $M' = M + \Delta$. Then we can fire σ infinitely many times, i.e. $M \overset{\sigma}{\Longrightarrow} M' + \Delta \overset{\sigma}{\Longrightarrow} M' + 2\Delta \overset{\sigma}{\Longrightarrow} \cdots$, hence $\mathcal{R}(\mathcal{N}, M)$ is infinite.
2. By the definition of $Supp(p)$, for all $\nu \in Supp(p)$, there exists a marking such that $[(i, \mathfrak{b})] \overset{*}{\Longrightarrow} M$ with $(p, \nu) \in M$. Since $Supp(p)$ is infinite, $\mathcal{R}(\mathcal{N}, [(i, \mathfrak{b})])$ is infinite as well. □

Lemma 4.7. *Let $\mathcal{N} = \langle P, T, A, \mathcal{E}, g, l \rangle$ be an adaptive net and $p \in P$. Then $Supp(p)$ is an infinite set iff at least one of the following conditions holds:*

1. *There exists a marking $M \in \mathcal{R}(\mathcal{N}, [(i, \mathfrak{b})])$ and a net token $(p, (N, M_N)) \in M$ such that (N, M_N) is unbounded.*
2. *There is no $\max_{(N,M) \in Supp(p)} size(N)$.*

Proof. ($\Rightarrow$) We will prove that if both conditions do not hold then the net is unbounded. The facts that there exists no marking $M \in \mathcal{R}(\mathcal{N}, [(i, \mathfrak{b})])$ containing a net token $(p, (N, M_N)) \in M$ such that (N, M_N) is unbounded, and that for all $p \in P$, $\max_{(N,M) \in Supp(p)} size(N) \leq n$ for some $n \in \mathbb{N}$, implies that there is a finite number of token nets present in reachable markings. Therefore the set $Supp(p)$ is finite.

($\Leftarrow$) We have two cases:

1. Let $M \in \mathcal{R}(\mathcal{N}, [(i, \mathfrak{b})])$ and $(p, (N, M_N)) \in M$ such that (N, M_N) is unbounded. Then $\mathcal{R}(N, M_N)$ is infinite and for all $M'_N \in \mathcal{R}(N, M_N)$, $M \overset{\tau^*}{\Longrightarrow} M'$ such that $(p, (N, M'_N)) \in M'$. Hence $Supp(p)$ is infinite.
2. Since there is no $\max_{(N,M) \in Supp(p)} size(N)$, there exists an infinite sequence of token nets $(N_1, M_1), (N_2, M_2) \ldots \in Supp(p)$ so that $size(N_1) < size(N_2) < \cdots$. Hence $Supp(p)$ is infinite. □

Figure 13 summarizes the sources of unboundedness of an adaptive net:

- Figure 13(a) shows an unbounded adaptive workflow net where the marking can grow infinitely (Lemma 4.6.1) since

$$[(p, (N, [(i, \mathfrak{b})]))] < [(p, (N, [(i, \mathfrak{b})]))] + [(p', (N, [(i, \mathfrak{b})]))].$$

- Figure 13(b) shows an unbounded adaptive workflow net where the token net N is unbounded which corresponds to Lemma 4.7.1.
- Place p of net $\mathcal{N}$ in Figure 13 (c) and (d) has the support $Supp(p) = \bigcup_{n \in \mathbb{N}^+} (N_n, M_n)$, where N_n is

$$\underbrace{N || \cdots || N}_{n \text{ times}}$$

and M_n is a marking of N_n. Hence there is no $\max_{(N,M) \in Supp(p)} size(N)$ (Lemma 4.7.2).

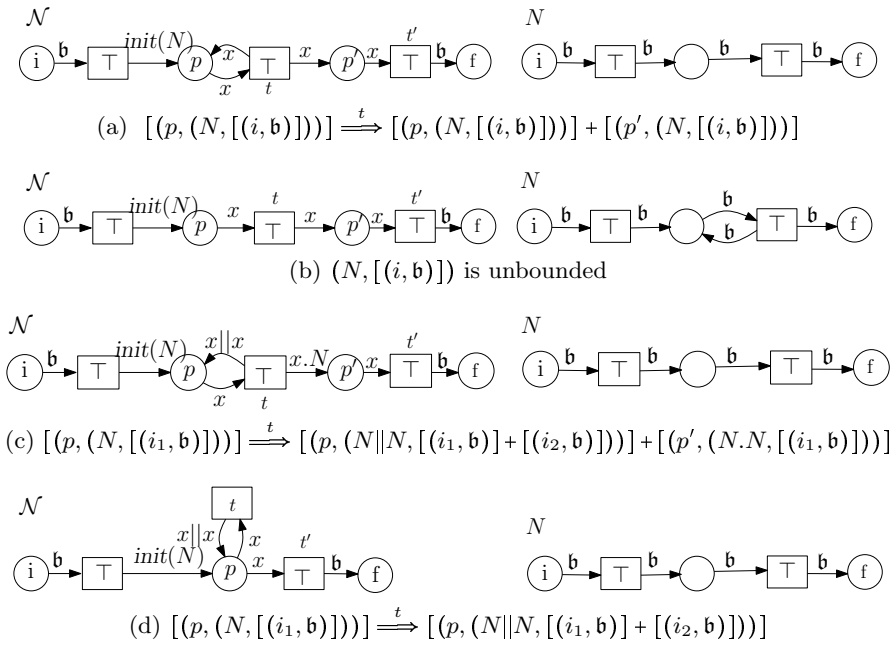


Fig. 13: Sources of unboundedness for adaptive workflow nets

- In Figure 13(c), by firing t, the number of tokens strictly grows, i.e. the markings $M = [(p,(N,[(i_1,\mathfrak{b})]))]$ and $M' = [(p,(N||N,[(i_1,\mathfrak{b})] + [(i_2,\mathfrak{b})]))] + [(p',(N.N,[(i,\mathfrak{b})]))]$ satisfy
  - for all $p \in P$, $\sum_{(p,\nu)\in M} M(p,\nu) \leq \sum_{(p,\nu)\in M'} M'(p,\nu)$
  - $\sum_{(p',\nu)\in M} M(p',\nu) < \sum_{(p',\nu)\in M'} M'(p',\nu)$.
- In Figure 13(d), by firing t, the number of tokens remains the same, i.e. $\sum_{(p,\nu)\in M} M(p,\nu) = \sum_{(p,\nu)\in M'} M'(p,\nu)$ for all $p \in P$, where $M = [(p,(N,[(i_1,\mathfrak{b})])]$ and $M' = [(p,(N||N,[(i_1,\mathfrak{b})] + [(i_2,\mathfrak{b})]))]$.

Note that the marking of the growing token net in p also is strictly bigger in both cases ((c) and (d)): $[(i_1,\mathfrak{b})] < [(i_1,\mathfrak{b})] + [(i_2,\mathfrak{b})]$, where i_1 is the initial place of N and i_2 is the initial place of the second copy of N in the net expression $N||N$.

Circumspectness

Untimely handling of exceptions can cause problems in real systems. We define a property called *circumspectness* that concerns prompt synchronization of final and exception transitions. Circumspectness means proper handling of exceptions: every exception transition enabled in lower level net can be readily synchronized with the corresponding transition in the upper level net.

Definition 4.18. [Circumspectness]
All nets from $\mathfrak{N}_1$ are defined to be circumspect. An adaptive workflow net $\mathcal{N}$ =

$\langle P, T, A, \mathcal{E}, g, l\rangle$ *of level* $k \geq 2$ *is called* circumspect *if for any marking* M *of* $\mathcal{N}$ *such that* $[(i, \mathfrak{b})] \stackrel{\sigma}{\Longrightarrow} M$*, for some transition sequence* $\sigma \in T_n^*$*, any place* p *and any marked token net* $(N, M_N) \in M(p)$ *on place* p*,* N *is circumspect and* $M_N \xrightarrow{t'}$ *for some exception transition* t' *in* N *with* $l(t') = e$ *implies that there exist* $t \in T$ *and* $b \in \mathcal{B}$ *such that* $M \xrightarrow{(t,b)}$ *in* $\mathcal{N}$*,* $b(\mathcal{E}(p,t)) = (N, M_N)$ *and* $g(t) = e(\mathcal{E}(p,t))$*.*

Observe that if the circumspectness condition is violated, the system net will ignore some exceptions of the token net.

4.4 Checking Properties of Adaptive Nets

In contrast to classical WF nets, boundedness is not a necessary condition of soundness for adaptive workflow nets. Since the net tokens of an adaptive workflow nets can become infinite, the set of reachable markings can have an unbounded size. To reduce the verification of soundness and circumspectness to a finite problem, we introduce an abstraction that replaces every net token in the adaptive workflow net by a colored token with the set of exceptions of the net token as a color. An adaptive workflow net is thus abstracted into a colored exception workflow net.

We start this section by presenting colored EWF nets and we then discuss our abstraction function.

4.4.1 Abstraction

Colored EWF nets We define colored EWF nets with a simple color set $\mathcal{C} = 2^{\Sigma^e} \cup \{\mathfrak{b}\}$, namely finite subsets of some set Σ^e of labels and a black token. We define a set *ColVar* of variables, set of constants $\mathit{ColCon} \in 2^{\Sigma^e}$ and a set of color expressions *ColExp* such that for $\eta \in \mathit{ColExp}$, $v \in \mathit{ColVar}$ and *ColCon*:

$$\eta = c \mid v \mid \eta \cup \eta.$$

We define expressions γ of our guard language *GL* by

$$\gamma = \top \mid r \in v,$$

where $r \in \Sigma^e$ and $v \in \mathit{Var}$. We now define our colored EWF nets as a simplified version of Definition 2.33.

Definition 4.19. [Colored EWF net]
A colored EWF net *is a tuple* $N = \langle P, T, A, \mathcal{E}, g, l\rangle$*, where* $\langle P, T, A, l\rangle$ *is an EWF net,* $\mathcal{E}: A \to \mathit{ColExp} \cup \{\mathfrak{b}\}$ *is an* arc expressions function *and* $g: T \to GL$ *is a* guard function*, both satisfying the same restrictions as in Definition 4.11.*

A binding is a mapping $b : Var \to \mathcal{C}$. We apply b to expressions and guards as follows:

$$b(e) = \begin{cases} c & \text{if } e = c \in ColCon; \\ \mathfrak{b} & \text{if } e = \mathfrak{b}; \\ b(e_1) \cup b(e_2) & \text{if } e = e_1 \cup e_2, e_1, e_2 \in ColExp; \\ \top & \text{if } e = \top; \\ r \in b(v) & \text{if } e = r \in v, v \in ColVar, r \in \Sigma^e. \end{cases}$$

A colored EWF net $N = \langle P, T, A, \mathcal{E}, g, l \rangle$ has markings that are multisets over $\mathbb{N}^{P \times \mathcal{C}}$. Like for adaptive nets, we denote the bag of tokens in place $p \in P$ in a marking M by $M(p)$, i.e. $M(p) \stackrel{\text{def}}{=} \sum_{(p,\nu) \in M} [\nu]$. Let M be a marking, $t \in T$ and $b \in \mathcal{B}$ such that $b(\mathcal{E}(p,t)) \in M(p)$ for all $p \in {}^\bullet t$ and $b(g(t))$ holds. Then t can fire in M with binding b:

$$M \xrightarrow{(t,b)} M - \sum_{p \in {}^\bullet t} [(p, b(\mathcal{E}(p,t)))] + \sum_{p \in t^\bullet} [(p, b(\mathcal{E}(t,p)))].$$

We write $M \xrightarrow{t} M'$ iff there exists a binding b such that $M \xrightarrow{(t,b)} M'$ and $M \xrightarrow{t}$ if there exists M such that $M \xrightarrow{(t,b)} M'$ and $M \xrightarrow{\sigma} M'$ if $M = M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \cdots \xrightarrow{t_{n-1}} M_n = M'$ and $\sigma = t_1 \cdots t_{n-1}$, for $n \geq 1$.

Abstraction function Here we present a mapping that relates adaptive workflow nets and colored EWF nets.

First we define the abstraction functions for arc expressions $\alpha_{\mathfrak{e}}$ that maps net (token) expressions to sets of exception labels from and for guards α_g respectively:

$$\alpha_{\mathfrak{e}}(e) = \begin{cases} \mathfrak{b} & \text{if } e = \mathfrak{b}; \\ v & \text{if } e = v \in Var; \\ \{l(t) | t \in T_e\} & \text{if } e = c \in Con, \mathcal{L}(c) = \langle P, T_n \cup T_e, A, \mathcal{E}, g, l \rangle; \\ \alpha_{\mathfrak{e}}(e_1) \cup \alpha_{\mathfrak{e}}(e_2) & \text{if } e = e_1 \ \rho \ e_2, e_1, e_2 \in CExpr, \rho \in \{\|, ., +\}; \\ \alpha_{\mathfrak{e}}(e') & \text{if } e = init(e'), e' \in Expr; \end{cases}$$

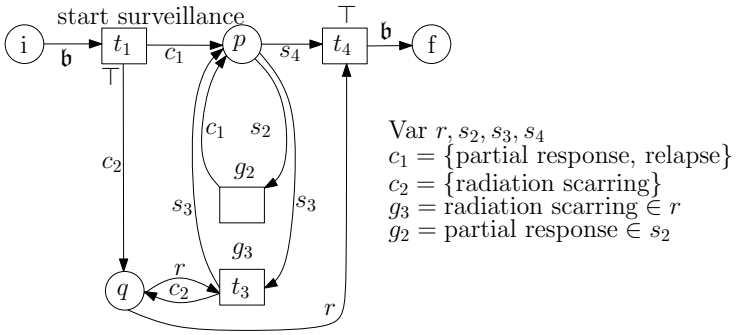


Fig. 14: Abstraction of the adaptive net in Figure 8

$$\alpha_g(g) = \begin{cases} r \in v & \text{if } g = r(v), v \in Var, r \in \{l'(t) | t \in T'_e \wedge \\ & \wedge \langle P', T'_e \cup T'_n, A', \mathcal{E}', g', l' \rangle \in \bigcup_{(t,p)\in A} \bigcup_{c \in Con(\mathcal{E}(t,p))} \mathcal{L}(c)\}; \\ \top & \text{if } g = \top. \end{cases}$$

Definition 4.20. [Abstraction]
The abstraction of an EWF net $N \in \mathfrak{N}_1$ *is* $\alpha_N(N) \stackrel{\text{def}}{=} N$. *The* abstraction *of an adaptive workflow net* $\mathcal{N} = \langle P, T, A, \mathcal{E}, g, l \rangle$ *of level* $k \geq 2$ *is* $\alpha_N(\mathcal{N}) = \langle P, T, A, \mathcal{E}^\alpha, g^\alpha, l \rangle$, $\mathcal{E}^\alpha(a) = \alpha_{\mathfrak{e}}(\mathcal{E}(a))$ *for all* $a \in A$ *and* $g^\alpha(t) = \alpha_g(g(t))$ *for all* $t \in T$.

Proposition 4.3. *Let* $\mathcal{N} = \langle P, T, A, \mathcal{E}, g, l \rangle$ *be an adaptive workflow net. Then* $\alpha_N(\mathcal{N})$ *is a colored EWF net.*

Proof. We can apply Definition 4.19 for $TExpr \subset ColExp$, and $\mathcal{C} = \{\mathfrak{b}\} \cup 2^{\Sigma^e}$, where $\Sigma^e = \{l'(t) | t \in T'_e \wedge \langle P', T'_e \cup T'_n, A', \mathcal{E}', g', l' \rangle \in \bigcup_{a \in A} \bigcup_{c \in Con(\mathcal{E}(a))} \mathcal{L}(c)\}$. From Definitions 4.11, 4.20 and 4.19, it follows that $\alpha_N(\mathcal{N})$ is a colored EWF net. □

The nets in Figures 14 and Figure 15 are abstractions of the adaptive workflow nets from Figures 8 and Figure 9, respectively. Note that the expressions on the input arcs of transitions contain either variables or the black token, so they coincide with the expressions on the arcs of the original net. The expressions on the output arcs of transition t_1 contain expressions on library nets and therefore they are mapped to the unions of the exception sets of these nets. For instance, the expression $init(Radiotherapy.STests\langle partial\ response, relapse\rangle_e)$ on the arc from t_1 to p is abstracted to the union of the exception set of $STests\langle partial\ response, relapse\rangle_e$, which is $\{partial\ response,\ relapse\}$, and the exception set of $STests\langle partial\ response, relapse\rangle_e$, which is the empty set. The expression $s_3 || init(Prophylactic)$ on the arc from t_3 to p is abstracted to the

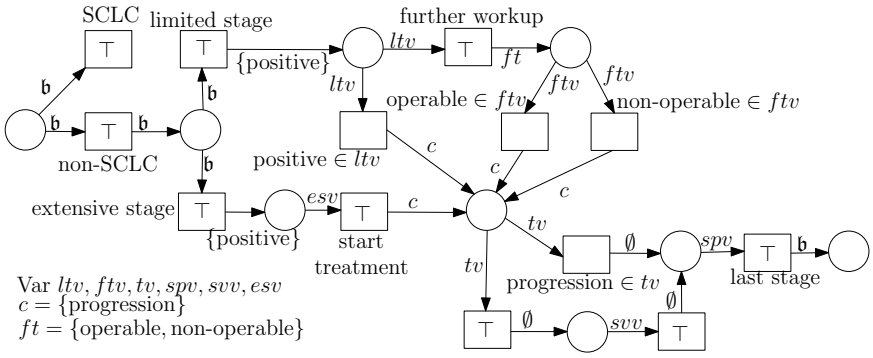


Fig. 15: Abstraction of the main SCLC protocol (Figure 9)

union of s_3 (since s_3 is a variable) and the exception set of *Prophylactic*, which is the empty set. The guard $g_2 = \textit{radiation scarring}(r)$ of transition t_3 is abstracted to $\textit{radiation scarring} \in r$, following the definition of the guard abstraction. The guard $\textit{final}(s_4)$ of transition t_4 is abstracted to $\top$.

Now we define the abstraction of a marking of an adaptive workflow net. Since our goal is checking soundness and circumspectness, we are particularly interested in the sets of exceptions of net tokens. Therefore, the abstraction of tokens is a function $\alpha_\nu : \mathfrak{M} \cup \{\mathfrak{b}\} \to \mathcal{C}$ defined as $\alpha_\nu(\mathfrak{b}) \stackrel{\text{def}}{=} \mathfrak{b}$ and $\alpha_\nu(\langle P, T_n \cup T_e, F, \mathcal{E}, g, l\rangle, M) \stackrel{\text{def}}{=} \{l(t) | t \in T_e\}$ for marked nets from $\mathfrak{M}$. The abstraction of a marking M of a marked net $(\mathcal{N}, M) \in \mathfrak{M}_k$, $k \geq 1$ is defined as follows: for all $p \in P, c \in \mathcal{C}$, $\alpha_M(M)(p,c) \stackrel{\text{def}}{=} \sum_{\nu \in \mathfrak{M}_{k-1} \wedge \alpha_\nu(\nu) = c} M(p, \nu)$.

Similarly, for a binding $b \in \mathcal{B}$, its abstraction $\alpha_\mathfrak{b}(b)$ is defined as follows:

$$\alpha_\mathfrak{b}(b)(e) = \begin{cases} c & e = c \in \mathit{Con}; \\ \mathfrak{b} & \text{if } e = \mathfrak{b}; \\ \alpha_\nu(b(v)) & \text{if } e = v; \\ \alpha_\mathfrak{b}(b)(\eta_1) \ \cup \ \alpha_\mathfrak{b}(b)(\eta_2) & \text{if } e = \eta_1 \cup \eta_2, \eta_1, \eta_2 \in \mathit{ColExp}. \end{cases}$$

$\mathcal{B}^\alpha$ is the set of all abstract bindings.

Proposition 4.4. *Let $e \in \mathit{Expr} \cup \mathit{TExpr}$, $g \in \mathcal{G}$ and $b \in \mathcal{B}$. Then, $\alpha_\nu(b(e)) = \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(e))$ and if $b(g)$ holds then $\alpha_\mathfrak{b}(b)(\alpha_g(g))$ holds.*

Proof. We proceed by structural induction on e and g, respectively. Let $e_1, e_2 \in \mathit{Expr}$, $te_1, te_2 \in \mathit{TExpr}$, $v \in \mathit{Var}$ and $r \in \Sigma_e$.

$e = \mathfrak{b}$ $\alpha_\nu(b(\mathfrak{b})) = \alpha_\nu(\mathfrak{b}) = \mathfrak{b} = \alpha_\mathfrak{b}(b)(\mathfrak{b})$.

$e = v$ $\alpha_\nu(b(v)) = \alpha_\mathfrak{b}(b)(v)$ follows from the definition.

$e = \textbf{\textit{init}}(e_1)$ Since $e_1 \in \mathit{Expr}$, we have $\alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(\mathit{init}(e_1))) = \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(e_1)) = \alpha_\mathfrak{e}(e_1) = \alpha_\mathfrak{e}(\mathit{init}(e_1)) = \alpha_\nu(\mathit{init}(e_1)) = \alpha_\nu(b(\mathit{init}(e_1)))$.

$e = te_1 \ . \ e_2$ We have $\alpha_\nu(b(te_1.e_2)) = \{l(t) | t \in T_e^1 \cup T_e^2 \wedge b(te_1) = = (\langle P_1, T_e^1 \cup T_n^1, A_1, \mathcal{E}_1, g_1, l_1\rangle, M) \wedge b(e_2) = \langle P_2, T_e^2 \cup T_n^2, A_2, \mathcal{E}_2, g_2, l_2\rangle\} = \{l(t) | t \in T_e^1 \wedge b(te_1) = (\langle P_1, T_e^1 \cup T_n^1, A_1, \mathcal{E}_1, g_1, l_1\rangle, M)\} \cup \{l(t) | t \in T_e^2 \wedge b(e_2) = \langle P_2, T_e^2 \cup T_n^2, A_2, \mathcal{E}_2, g_2, l_2\rangle\} = \alpha_\nu(b(te_1)) \cup \alpha_\mathfrak{e}(e_2)$. From the hypothesis, we have that $\alpha_\nu(b(e_1)) = \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(e_1))$ and $\alpha_\mathfrak{e}(e_2) = \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(e_2))$. Hence, we have $\alpha_\nu(b(te_1.e_2)) = \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(te_1)) \cup \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(e_2))$
$= \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(te_1) \cup \alpha_\mathfrak{e}(e_2)) = \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(te_1 \ . \ e_2))$.

$e = te_1 \parallel te_2$ We have $\alpha_\nu(b(te_1 \| te_2)) = \{l(t) | t \in T_e^1 \cup T_e^2 \wedge b(te_1) = (\langle P_1, T_e^1 \cup T_n^1, A_1, \mathcal{E}_1, g_1, l_1\rangle, M_1) \wedge b(e_2) = (\langle P_2, T_e^2 \cup T_n^2, A_2, \mathcal{E}_2, g_2, l_2\rangle, M_2)\} = \{l(t) | t \in T_e^1 \wedge b(te_1) = (\langle P_1, T_e^1 \cup T_n^1, A_1, \mathcal{E}_1, g_1, l_1\rangle, M_1)\} \cup \{l(t) | t \in T_e^2 \wedge b(te_2) = \{l(t) | t \in T_e^2 \cup b(te_2) = (\langle P_2, T_e^2 \cup T_n^2, A_2, \mathcal{E}_2, g_2, l_2\rangle, M_2)\} = \alpha_\nu(b(te_1)) \cup \alpha_\mathfrak{e}(b(te_2))$.
Since $\alpha_\nu(b(te_1)) = \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(te_1))$ and $\alpha_\nu(b(te_2)) = \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(te_2))$, we have that $\alpha_\nu(b(te_1 \| te_2)) = \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(te_1)) \cup \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(te_2)) = \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(te_1) \cup \alpha_\mathfrak{e}(te_2)) = \alpha_\mathfrak{b}(b)(\alpha_\mathfrak{e}(te_1 \parallel te_2))$.

$g = \top$ Since $\alpha_\mathfrak{b}(b)(\alpha_g(g)) = \alpha_\mathfrak{b}(b)(\top) = \top$, $\alpha_\mathfrak{b}(b)(\alpha_g(g))$ holds.

$g = r(v)$ We have $\alpha_\mathfrak{b}(b)(\alpha_g(r(v))) = \alpha_\mathfrak{b}(b)(r \in v) = r \in \alpha_\mathfrak{b}(b)(v)$. Moreover $r \in \alpha_\mathfrak{b}(b)(v) = r \in \alpha_\nu(b(v)) = \alpha_g(r(b(v))) = \alpha_g(b(r(v)))$. Since $b(r(v))$ holds, $\alpha_g(b(r(v)))$ holds as well, hence $\alpha_\mathfrak{b}(b)(\alpha_g(g))$ holds.

□

In the rest of the subsection we consider adaptive nets of level $k \geq 2$.

Firings of the abstraction of an adaptive net enjoy a close correspondence with firings of the adaptive net. First of all, every firing in the adaptive net can be weakly simulated in the abstraction.

Lemma 4.8. *Let $M \overset{\sigma}{\Longrightarrow} M'$ be a firing in an adaptive net $\mathcal{N}$. Then, in $\alpha_N(\mathcal{N})$, we have $\alpha_M(M) \overset{\sigma}{\longrightarrow} \alpha_M(M')$.*

Proof. Let $M \overset{t}{\Longrightarrow} M'$. Then there is a binding $b \in \mathcal{B}$ such that $M \overset{(t,b)}{\Longrightarrow} M'$, i.e. $b(\mathcal{E}(p,t)) \in M(p)$ for all $p \in {}^\bullet t$, $b(g(t))$ holds and

$$M' = M - \sum_{p \in {}^\bullet t} [(p, b(\mathcal{E}(p,t)))] + \sum_{p \in t^\bullet} [(p, b(\mathcal{E}(t,p)))].$$

By applying the abstraction function (Definition 4.20), we have that for each $p \in {}^\bullet t$, $\alpha_\nu(b(\mathcal{E}(p,t))) \in \alpha_M(M)(p)$. By Proposition 4.4, we have

$$\alpha_\mathfrak{b}(b)(\mathcal{E}^\alpha(p,t)) \in \alpha_M(M)(p)$$

for all $p \in {}^\bullet t$. Since $b(g(t))$ holds, by Proposition 4.4, we have that $\alpha_{\mathfrak{b}}(b)(g^\alpha(t))$ holds as well. Hence, there exists M_α so that $\alpha_M(M) \xrightarrow{(t,b)} M_\alpha$ and

$$M_\alpha = \alpha_M(M) - \sum_{p \in {}^\bullet t} [(p, \alpha_{\mathfrak{b}}(b)(\mathcal{E}^\alpha(p,t)))] + \sum_{p \in t^\bullet} [(p, \alpha_{\mathfrak{b}}(b)(\mathcal{E}^\alpha(t,p)))].$$

By Proposition 4.4,

$$M_\alpha = \alpha_M(M) - \sum_{p \in {}^\bullet t} [(p, \alpha_\nu(b(\mathcal{E}(p,t))))] + \sum_{p \in t^\bullet} [(p, \alpha_\nu(b(\mathcal{E}(t,p))))] = \alpha_M(M').$$

Thus $\alpha_M(M) \xrightarrow{(t,\alpha_{\mathfrak{b}}(b))} \alpha_M(M')$, i.e. $\alpha_M(M) \xrightarrow{t} \alpha_M(M')$. By applying induction on the length of σ, we conclude that $M \overset{\sigma}{\Longrightarrow} M'$ implies $\alpha_M(M) \xrightarrow{\sigma} \alpha_M(M')$. □

Second, every two markings connected by a firing sequence in the abstract net are abstractions of some markings of the adaptive net connected by a firing sequence.

Lemma 4.9. *Let $\mathcal{N} = \langle P, T, A, \mathcal{E}, g, l\rangle$ be an adaptive net, such that for all $c \in \bigcup_{(t,p)\in A} Con(\mathcal{E}(t,p))$, c is (relaxed/weakly) sound, and $\alpha_N(\mathcal{N})$ its abstraction.*

(1) Given a marking M of $\mathcal{N}$, with all its net tokens being in the initial marking, and $M_\alpha = \alpha_M(M)$ of $\alpha_N(\mathcal{N})$, any firing sequence $M_\alpha \xrightarrow{\sigma} M'_\alpha$ can be simulated by $M \overset{\sigma}{\Longrightarrow} M'$ so that $\alpha_M(M') = M'_\alpha$ and all net tokens in M' are in the initial marking.

(2) Given a marking M of $\mathcal{N}$, with all net tokens being in the final marking, and $M_\alpha = \alpha_M(M)$ of $\alpha_N(\mathcal{N})$, any firing sequence $M_\alpha \xrightarrow{\sigma} M'_\alpha$ with $g(t) = \top$ for all $t \in \sigma$ can be simulated by $M \overset{\sigma}{\Longrightarrow} M'$ so that $\alpha_M(M') = M'_\alpha$ and all net tokens in M' are in the final marking.

Proof. (1) We use induction on the length of the transition sequence σ. For $\sigma = \epsilon$, the statement trivially holds with $M' = M$. Suppose that the statement holds for every firing sequence $M_\alpha \xrightarrow{\sigma} M'_\alpha$ with $\sigma \in T_n^k$, for some $k \geq 0$.

Consider M'_α such that $M_\alpha \xrightarrow{\sigma t} M'_\alpha$ for some $\sigma t \in T_n^{k+1}$, then $M_\alpha \xrightarrow{\sigma} M''_\alpha \xrightarrow{t} M'_\alpha$, for some M''_α. By the induction hypothesis, $M \overset{\sigma}{\Longrightarrow} M''$, for some M'' such that all its net tokens have the initial marking and $\alpha_M(M'') = M''_\alpha$.

Since $M''_\alpha \xrightarrow{t} M'_\alpha$, there exists a binding $b^\alpha \in \mathcal{B}^\alpha$ such that $M''_\alpha \xrightarrow{(t,b)} M'_\alpha$, i.e. $b^\alpha(\mathcal{E}^\alpha(p,t)) \in M''_\alpha(p)$, for all $p \in {}^\bullet t$, $b^\alpha(g^\alpha(t))$ holds and

$$M'_\alpha = M''_\alpha - \sum_{p \in {}^\bullet t} [(p, b^\alpha(\mathcal{E}^\alpha(p,t)))] + \sum_{p \in t^\bullet} [(p, b^\alpha(\mathcal{E}^\alpha(t,p)))].$$

By the definition of abstraction, there is a binding $b \in \mathcal{B}$ such that $b^\alpha = \alpha_{\mathfrak{b}}(b)$, for all $v \in \bigcup_{(p,t)\in F} Var(\mathcal{E}(p,t))$, $b(v) = (N, [(i, \mathfrak{b})])$, for some N, and $b(\mathcal{E}(p,t)) \in M''(p)$, for all $p \in {}^\bullet t$. We have the following cases of guards in $\mathcal{N}$:

(*a*) $g(t) = \mathit{final}(v)$. Let $M'' \xrightarrow{\tau^*} M'''$ be the firing by which $b(v) = (N, M_0)$ reaches the final marking (by Lemma 4.4, by Lemma 4.5 and the fact that all the constants in the net expressions are (relaxed/weak) sound). Let b' be the binding which coincides with b, except for $b'(v) = (N, [(f, \mathfrak{b})])$.

(*b*) $g(t) = e(v)$. Let $M'' \xrightarrow{\tau^*} M'''$ be the firing so that in $b(v) = (N, [(i, \mathfrak{b})])$ reaches $M_N \xrightarrow{t'}$ and $l(t') = e$ (all net tokens are (relaxed/weak) sound, hence t' is quasi-live in $b(v)$). Let b' be the binding which coincides with b, except for $b'(v) = (N, M_N)$.

(*c*) $g(t) = \top$. Let $b' = b$ and $M''' = M''$.

Then $b'(g(t))$ holds and $M''' \overset{(t,b')}{\Longrightarrow} M'$, where $M' = M''' - \sum_{p \in {}^\bullet t}[(p, b'(\mathcal{E}(p,t)))] + \sum_{p \in t^\bullet}[(p, b'(\mathcal{E}(t,p)))]$ and $\alpha_M(M''') = \alpha_M(M'')$.

By Proposition 4.4, we have

$$\begin{aligned}\alpha_M(M') &= \alpha_M(M''') - \sum_{p \in {}^\bullet t}[(p, \alpha_\nu(b'(\mathcal{E}(p,t))))] + \sum_{p \in t^\bullet}[(p, \alpha_\nu(b'(\mathcal{E}(t,p))))] \\ &= M''_\alpha - \sum_{p \in {}^\bullet t}[(p, b^\alpha(\mathcal{E}^\alpha(p,t)))] + \sum_{p \in t^\bullet}[(p, b^\alpha(\mathcal{E}^\alpha(t,p)))] = M'_\alpha\end{aligned}$$

and all net tokens in M' except the ones have the initial marking.

(2) We use induction on the length of the transition sequence σ. For $\sigma = \epsilon$, the statement trivially holds with $M' = M$. Suppose that the statement holds for every firing sequence $M_\alpha \xrightarrow{\sigma} M'_\alpha$ with $\sigma \in T_n^k$, for some $k \geq 0$.

Consider M'_α such that $M_\alpha \xrightarrow{\sigma t} M'_\alpha$ for some $\sigma t \in T_n^{k+1}$, then $M_\alpha \xrightarrow{\sigma} M''_\alpha \xrightarrow{t} M'_\alpha$, for some M''_α. By the induction hypothesis, $M \overset{\sigma}{\Longrightarrow} M''$, for some M'' such that all its net tokens have the final marking and $\alpha_M(M'') = M''_\alpha$.

Since $M''_\alpha \xrightarrow{t} M'_\alpha$, there exists a binding $b^\alpha \in \mathcal{B}^\alpha$ such that $M''_\alpha \xrightarrow{(t,b)} M'_\alpha$, i.e. $b^\alpha(\mathcal{E}^\alpha(p,t)) \in M''_\alpha(p)$, for all $p \in {}^\bullet t$, $b^\alpha(g^\alpha(t))$ holds and

$$M'_\alpha = M''_\alpha - \sum_{p \in {}^\bullet t}[(p, b^\alpha(\mathcal{E}^\alpha(p,t)))] + \sum_{p \in t^\bullet}[(p, b^\alpha(\mathcal{E}^\alpha(t,p)))].$$

By the definition of abstraction, there is a binding $b \in \mathcal{B}$ such that $b^\alpha = \alpha_{\mathfrak{b}}(b)$, for all $v \in \bigcup_{(p,t) \in A} \mathit{Var}(\mathcal{E}(p,t))$, $b(v) = (N, [(f, \mathfrak{b})])$, for some N, and $b(\mathcal{E}(p,t)) \in M''(p)$, for all $p \in {}^\bullet t$.

Since the all net tokens have the final marking, in either cases $g(t) = \mathit{final}(v)$ or $g(t) = \top$, $b(g(t))$ holds. Hence $M'' \overset{(t,b)}{\Longrightarrow} M'''$, where

$$M''' = M'' - \sum_{p \in {}^\bullet t}[(p, b(\mathcal{E}(p,t)))] + \sum_{p \in t^\bullet}[(p, b(\mathcal{E}(t,p)))].$$

Let $M''' \xrightarrow{\tau^*} M'$ be the firing by which all $(N, M_N) \in \bigcup_{(t,p) \in A} b(\mathcal{E}(t,p))$ reach the final marking. Let b' be the binding such that $b'(v) = b(v)$ if $b(v) =$

$(N, [(f, \mathfrak{b})])$ and $b'(v) = (N, [(f, \mathfrak{b})])$ if $b(v) = (N, M_N)$ with $M_N \neq [(f, \mathfrak{b})]$. Hence $\alpha_M(M''') = \alpha_M(M')$ and by Proposition 4.4, we have

$$\begin{aligned}\alpha_M(M') &= \alpha_M(M'') - \sum_{p \in {}^\bullet t} [(p, \alpha_\nu(b(\mathcal{E}(p,t))))] + \sum_{p \in t^\bullet} [(p, \alpha_\nu(b(\mathcal{E}(t,p))))] \\ &= M''_\alpha - \sum_{p \in {}^\bullet t} [(p, b^\alpha(\mathcal{E}^\alpha(p,t)))] + \sum_{p \in t^\bullet} [(p, b^\alpha(\mathcal{E}^\alpha(t,p)))] = M'_\alpha\end{aligned}$$

and all net tokens in M' have the final marking. □

4.4.2 Checking Soundness and Circumspectness

Soundness Now we can reduce the check of soundness of an adaptive workflow net to the check of a behavioral property on its abstraction:

Theorem 4.1 (soundness). *Let $\mathcal{N}$ be an adaptive net and $\alpha_N(\mathcal{N})$ its abstraction, such that for all $c \in \bigcup_{(t,p)\in A} Con(\mathcal{E}(t,p))$, c is sound. Then $\mathcal{N}$ is sound iff for all abstract markings M_α reachable by firing of non-exception transitions, i.e. $[(i, \mathfrak{b})] \xrightarrow{\sigma} M_\alpha$ with $\sigma \in (T_n)^*$, $M_\alpha \xrightarrow{\sigma'} [(f, \mathfrak{b})]$ for some $\sigma' \in (T_n)^*$ such that $g^\alpha(t) = \top$ for all $t \in \sigma'$ and $\alpha(\mathcal{N})$ is quasi-live.*

Proof. ($\Rightarrow$) Let $[(i, \mathfrak{b})] \xrightarrow{\sigma} M_\alpha$ in $\alpha_N(\mathcal{N})$, with $\sigma \in T_n^*$. By Lemma 4.9.(1), there exists a marking M so that $[(i, \mathfrak{b})] \stackrel{\sigma}{\Longrightarrow} M$ in $\mathcal{N}$, with $\alpha_M(M) = M_\alpha$ and all net tokens in M have the initial marking. Since $\mathcal{N}$ is sound, $M' \stackrel{\gamma}{\Longrightarrow} [(f, \mathfrak{b})]$, $\gamma \in T_n^*$ and $g(t) \in \{\top, \mathit{final}(v)\}$, for all $t \in \gamma$. By Lemma 4.8, $\alpha_M(M') \xrightarrow{\gamma} [(f, \mathfrak{b})]$ in $\alpha_N(\mathcal{N})$, $\gamma \in T_n^*$ and $g^\alpha(t) = \top$ for all $t \in \gamma$.

Since $\mathcal{N}$ is sound, every transition t of $\mathcal{N}$ is quasi-live. Thus there is a reachable marking M of $\mathcal{N}$ such that $M \xrightarrow{t}$. By Lemma 4.8, $\alpha_M(M)$ is reachable in $\alpha_N(\mathcal{N})$ and $\alpha_M(M) \stackrel{t}{\Longrightarrow}$, hence t is quasi-live in $\alpha_N(\mathcal{N})$.

($\Leftarrow$) Let $[(i, \mathfrak{b})] \stackrel{\sigma}{\Longrightarrow} M$ in $\mathcal{N}$, where $\sigma \in T_n^*$. Since all net tokens are sound $M \xrightarrow{\tau^*} M'$ such that all net tokens of M' have the final marking. By Lemma 4.8, $[(i, \mathfrak{b})] \xrightarrow{\sigma} \alpha_M(M) = \alpha_M(M')$ in $\alpha_N(\mathcal{N})$ and by hypothesis, $\alpha_M(M) \xrightarrow{\sigma'} [(f, \mathfrak{b})]$ in $\alpha_N(\mathcal{N})$, where $\sigma, \sigma' \in T_n^*$ so that $g(t) = \top$, for all $t \in \sigma'$. By Lemma 4.9.(2), we have that $M' \stackrel{\sigma''}{\Longrightarrow} [(f, \mathfrak{b})]$ in $\mathcal{N}$ and all $t \in \gamma$ are in T_n and $g(t) \in \{\top, \mathit{final}(v)\}$.

Since $\alpha_N(\mathcal{N})$ is sound, every transition t is quasi-live. Hence, there exists M_α such that $[(i, \mathfrak{b})] \xrightarrow{\sigma} M_\alpha \xrightarrow{t}$, for some σ. By Lemma 4.9.(1), there exists a marking M of $\mathcal{N}$ reachable by non-exception transition such that all net tokens have the initial marking and $\alpha_M(M) = M_\alpha$. If $g(t) = e(v)$, then $M \stackrel{(t,b)}{\Longrightarrow}$, where $b(v) = (N, M_N)$, $l(t) = e$ and $M_N \xrightarrow{t}$; if $g(t) = \mathit{final}(v)$, since all net tokens are sound, $M \stackrel{(t,b)}{\Longrightarrow}$ and $b(v) = (N, [(f, \mathfrak{b})])$; if $g(t) = \top$, $M \xrightarrow{t}$, so $M \stackrel{t}{\Longrightarrow}$. Hence, t is quasi-live in $\mathcal{N}$. □

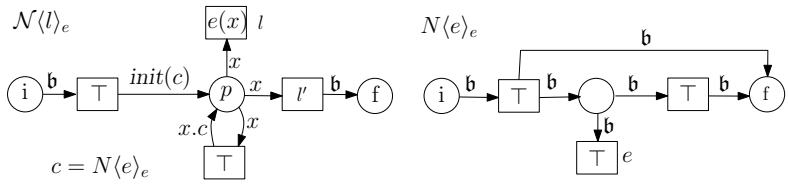


Fig. 16: Sound and circumspect adaptive net

Note that there exist sound adaptive nets for which the theorem cannot be applied. For instance, $\mathcal{N}\langle l\rangle_e$ in Figure 16 is sound even though the net $N\langle e\rangle_e$ is unsound and $\alpha_N(\mathcal{N}\langle l\rangle_e)$ satisfies the property that for all M_α with $[(i,\mathfrak{b})] \xrightarrow{\sigma} M_\alpha$, $M_\alpha \xrightarrow{\sigma'} [(f,\mathfrak{b})]$, for some $\sigma,\sigma' \in T_n^*$.

Figure 17 shows the abstractions of the unbounded adaptive nets in Figure 13. Note that when the number of tokens grows in some places (cases (a) and (c)), the abstraction is unbounded as well. However the boundednness of the abstraction does not imply the boundedness of the adaptive net (see cases (b) and (d) as counterexamples), since the abstraction ignores the net tokens and their sizes.

We now prove that the abstraction of a sound adaptive workflow net is bounded.

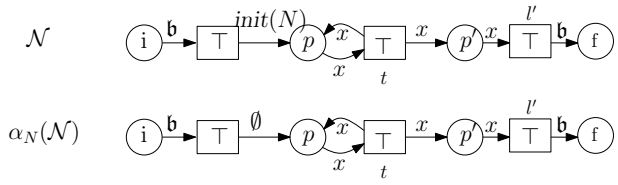
Lemma 4.10. *Let $\mathcal{N}$ be a sound adaptive net. Then, $\mathcal{R}(\alpha_N(\mathcal{N}),[(i,\mathfrak{b})])$ is finite.*

Proof. Suppose $\mathcal{R}(\alpha_N(\mathcal{N}),[(i,\mathfrak{b})])$ is infinite. Hence there exists an infinite sequence of pairwise distinct markings $M_\alpha^1, M_\alpha^2, \ldots$ reachable by non-exceptional transitions, i.e. $[(i,\mathfrak{b})] \xrightarrow{*} M_\alpha^1 \xrightarrow{*} M_\alpha^2 \xrightarrow{*} \cdots$. Since the set of colors $\mathcal{C}$ of $\alpha_N(\mathcal{N})$ is finite (Proposition 4.3), we can apply Dickson's Lemma (Lemma 2.1) w.r.t. the set of markings $\mathbb{N}^{P\times\mathcal{C}}$. Hence there exist an infinite subsequence $M_\alpha^{k_1}$, $M_\alpha^{k_2}, \ldots$ of $M_\alpha^1, M_\alpha^2, \ldots$, where $k_1 < k_2 < \cdots$, such that $M_\alpha^{k_i} = M_\alpha^{k_i+1} + \Delta_i$ where $\Delta_i > \varnothing$ for all $i \geq 1$. By Theorem 4.1, $M_\alpha^{k_1} \xrightarrow{\sigma} [(f,\mathfrak{b})]$ for some $\sigma \in T_n^*$. Then $M_\alpha^{k_2} \xrightarrow{\sigma} [(f,\mathfrak{b})]+\Delta_1$. Since f is a sink place, $[(f,\mathfrak{b})]+\Delta_1$ cannot reach $[(f,\mathfrak{b})]$, which contradicts Theorem 4.1. So $\mathcal{R}(\alpha_N(\mathcal{N}),[(i,\mathfrak{b})])$ is finite. □

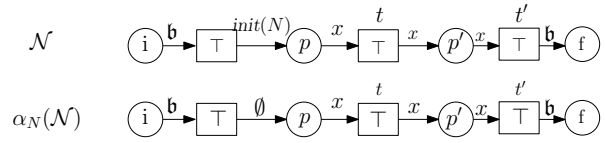
This lemma emphases the fact that the only source of unboundedness for sound adaptive nets is actually the growing size of the net tokens as pictured in Figure17.(d).

Theorem 4.2 (weak soundness). *Let $\mathcal{N}$ be an adaptive net and $\alpha_N(\mathcal{N})$ its abstraction, such that for all $c \in \bigcup_{(t,p)\in A} Con(\mathcal{E}(t,p))$, c is weakly sound. Then $\mathcal{N}$ is weakly sound iff its abstraction is quasi-live and there exists a firing sequence $[(i,\mathfrak{b})] \overset{\sigma}{\Longrightarrow} [(f,\mathfrak{b})]$ for some $\sigma' \in T_n^*$ such that $g^\alpha(t) = \top$ for all $t \in \sigma'$.*

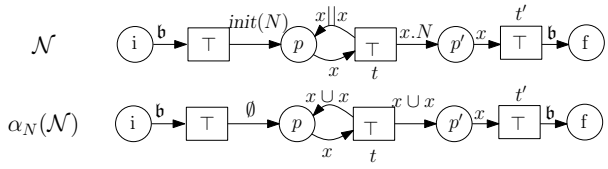
Proof. The proof follows from Lemma 4.8 and Lemma 4.9. □



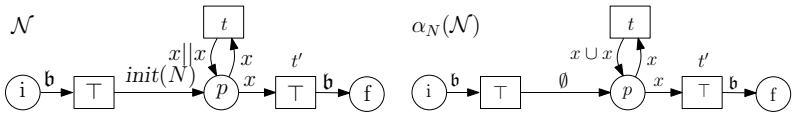
(a) $\alpha_N(\mathcal{N})$ is unbounded: $[(p,\varnothing)] \xRightarrow{t} [(p,\varnothing)] + [(p',\varnothing)]$



(b) $\alpha_N(\mathcal{N})$ is bounded



(c) $\alpha_N(\mathcal{N})$ is unbounded: $[(p,\varnothing)] \xRightarrow{t} [(p,\varnothing)] + [(p',\varnothing)]$



(d) $\alpha_N(\mathcal{N})$ is bounded

Fig. 17: Abstractions of unbounded adaptive nets

Theorem 4.3 (relaxed soundness). *Let $\mathcal{N}$ be an adaptive net and $\alpha_N(\mathcal{N})$ its abstraction, such that for all $c \in \bigcup_{(t,p)\in A} Con(\mathcal{E}(t,p))$, c is relaxed sound. Then $\mathcal{N}$ is relaxed sound iff its abstraction is relaxed sound, i.e $\alpha_N(\mathcal{N})$ is t-quasi-live for all $t \in T_e$ and for all transitions $t \in T_n$, there exist M_α, M'_α, $\sigma \in T_n^*$, $\sigma' \in T_n^*$ such that $g^\alpha(t) = \top$ for all $t \in \sigma'$ and $[(i,\mathfrak{b})] \xRightarrow{\sigma} M_\alpha \xRightarrow{t} M'_\alpha \xRightarrow{\sigma'} [(f,\mathfrak{b})]$.*

Proof. The proof follows from Lemma 4.8 and Lemma 4.9. □

Circumspectness As explained above, we would like to check circumspectness of an adaptive workflow net by checking the corresponding property of a colored EWF. The following theorem establishes the link between circumspectness of an adaptive net and a similar property of its abstraction.

Theorem 4.4 (circumspectness). *Let $\mathcal{N}$ be an adaptive workflow net of level $k \geq 2$ and $\alpha_N(\mathcal{N})$ its abstraction, such that for all $c \in \bigcup_{(t,p)\in A} Con(\mathcal{E}(t,p))$,*

c is circumspect and sound. $\mathcal{N}$ *is circumspect iff for any marking* M_α *of* $\alpha_N(\mathcal{N})$ *such that* $[(i,\mathfrak{b})] \xrightarrow{\sigma} M_\alpha$ *where* $\sigma \in T_n^*$*, for all tokens* $(p,c^\alpha) \in M_\alpha$ *and all exceptions* $r \in c^\alpha$*, there exist* $t \in T$*,* $b \in \mathcal{B}^\alpha$*,* $v \in Var$ *so that* $\mathcal{E}^\alpha(p,t) = v$*,* $b(v) = c^\alpha$*,* $g^\alpha(t) = r \in v$ *and* $M_\alpha \xrightarrow{(t,b)}$*.*

Proof. ($\Rightarrow$) Let M_α be a marking reachable by the firings of non-exception transitions in $\alpha_N(\mathcal{N})$, $(p,c^\alpha) \in M_\alpha$ and $r \in c^\alpha$. By Lemma 4.9.(1), there exists a marking M in $\mathcal{N}$ reachable by firings of non-exception transitions such that all tokens nets have the initial marking and $\alpha_M(M) = M_\alpha$. Hence, there is a token net $(N,[(i,\mathfrak{b})]) \in M(p)$ with the set of exception labels c^α and a transition t' of N such that $l(t') \in c^\alpha$. Since t' is quasi-live in N, $[(i,\mathfrak{b})] \xrightarrow{*} M_N \xrightarrow{t'}$. Hence $M \xrightarrow{\tau^*} M'$, so that $(N,M_N) \in M'(p)$ and $M_\alpha = \alpha_M(M')$. Since $\mathcal{N}$ is circumspect there exists a t of $\mathcal{N}$ such that $M' \xrightarrow{t}$ and $g(t) = r(v)$, where $v \in Var$. By Lemma 4.8, $\alpha_M(M') = M_\alpha \xrightarrow{t}$ and moreover $g^\alpha(t) = r \in v$.

($\Leftarrow$) Let M be a marking reachable by the firing of non-exception transitions in $\mathcal{N}$ such that there is a marked token net $(p,(N,M_N)) \in M$ and $M_N \xrightarrow{t'}$ for some exception transition t' in N and $l(t') = r$. By Lemma 4.8, there exists a marking M^α reachable by non-exception transitions in $\alpha_N(\mathcal{N})$ such that $\alpha_M(M) = M_\alpha$ and $r \in \bigcup_{(p,c^\alpha)\in M_\alpha} c^\alpha$. Hence there exists a t in $\mathcal{N}$ such that $g^\alpha(t) = r \in v$ and $M_\alpha \xrightarrow{t}$, i.e there exists a binding b^α, with $b^\alpha(\mathcal{E}^\alpha(p,t)) \in M_\alpha(p)$, for all $p \in {}^\bullet t$ and $b^\alpha(g^\alpha(t))$ holds. By the definition of abstraction there is a binding b with $\alpha_\mathfrak{b}(b) = b^\alpha$, $b(\mathcal{E}(p,t)) \in M(p)$, for all $p \in {}^\bullet t$, and $b(g(t)) = b(r(v))$ holds. Hence, $M \xrightarrow{t}$ in $\mathcal{N}$. □

Figure 16 shows a circumspect net, where the net token is not sound, however the net satisfies the condition in the above theorem on its abstraction.

Verification on abstraction We are interested in nets which are sound at all levels, including their net tokens. Therefore we introduce the notion of strong soundness and strong circumspectness.

Definition 4.21 (STRONG SOUNDNESS).
All sound nets from $\mathfrak{N}_1$ *are defined to be* strongly sound*. An adaptive net* $\mathcal{N} = \langle P,T,A,\mathcal{E},g,l\rangle$ *is* strongly sound *iff* $\mathcal{N}$ *is sound and all* $c \in \bigcup_{(t,p)\in A} Con(\mathcal{E}(t,p))$ *are strongly sound.*

Definition 4.22 (STRONG CIRCUMSPECTNESS).
All nets from $\mathfrak{N}_1$ *are defined to be* strongly circumspect*. An adaptive net* $\mathcal{N} = \langle P,T,A,\mathcal{E},g,l\rangle$ *is* strongly circumspect *iff* $\mathcal{N}$ *is circumspect and all* $c \in \bigcup_{(t,p)\in A} Con(\mathcal{E}(t,p))$ *are strongly sound and strongly circumspect.*

The check of strong soundness and strong circumspectness for an adaptive net N of level $k \geq 1$ can be performed either top-down or bottom-up. In the

Algorithm 4: *CheckStrongSound*($\mathcal{N}$)

Input: $\mathcal{N} = \langle P, T_n \cup T_e, A, \mathcal{E}, g, l \rangle$
Output: *true* if $\mathcal{N}$ is strongly sound; *false* if $\mathcal{N}$ is not strongly sound;

(a) **if** $\alpha_N(\mathcal{N})$ *is unbounded* **then**
  return *false*
else
  if $\neg((\mathcal{R}((\alpha_N(\mathcal{N}))_{|T_n}, [(i, \mathfrak{b})]) \subseteq \mathcal{S}((\alpha_N(\mathcal{N}))_{|T_n}, [(f, \mathfrak{b})])) \wedge$
  $\wedge\ (\forall t \in T_n \cup T_e \exists M \in \mathcal{R}_{|T_n}(\alpha_N(\mathcal{N}), [(i, \mathfrak{b})]) : M \xrightarrow{t}))$ **then**
    return false;

(b) **if** $\forall N \in \bigcup_{(t,p)\in A} Con(\mathcal{E}(t, p)) : CheckStrongSound(N) == true$ **then**
  return *true*
else
  return *false*;

Algorithm 5: *CheckStrongCirc*($\mathcal{N}$)

Input: $\mathcal{N} = \langle P, T_n \cup T_e, A, \mathcal{E}, g, l \rangle$
Output: *true* if $\mathcal{N}$ is strongly circumspect ; *false* if $\mathcal{N}$ is not strongly circumspect ;

(a) **forall** $M^\alpha \in \mathcal{R}((\alpha_N(\mathcal{N}))_{|T_n}, [(i, \mathfrak{b})]), p \in P$ **do**
  if $\exists (p, c^\alpha) \in M^\alpha, r \in c^\alpha : (\forall t' \in p^\bullet : \neg(g(t) == [l \in \mathcal{E}(p, t')]))$ **then**
    return *false*;

(b) **if** $\forall N \in \bigcup_{(t,p)\in A} Con(\mathcal{E}(t, p)) : CheckStrongCirc(N) \wedge CheckStrongSound(N)$ **then**
  return *true*
return *false*;

---

top-down approach, depicted in Algorithms 4 and 5, we check the sufficient conditions on the abstraction of N *assuming* the corresponding properties of the constants (line (a)). Next we proceed and consider strong soundness and strong circumspectness of the nets to which constants in the system net are mapped, *assuming* the corresponding properties of the constants appearing in these nets (line (b)). In the bottom-up case one first has to find all constants of level 1 appearing in the nets, analyze these nets and then proceed to the following level (lines (a) and (b) are reversed).

To analyze strong soundness and strong circumspectness of the adaptive net in Figure 8 we consider its abstraction in Figure 14. Clearly, for any markings reachable in Figure 14 there exists a sequence of non-exceptional guarded firings reaching $[(f, \mathfrak{b})]$. Therefore, the adaptive net in Figure 8 is strongly sound. The

net is not strongly circumspect, since in the abstract net there is no transition having the guard *relapse* $\in v$ while the exception label *relapse* is mentioned in $[(p, c_1)] + [(q, c_2)]$.

4.5 Related Work

Net in nets are extensively studied in the Petri net literature (see e.g [24, 81, 85, 94, 95, 101, 136]). The goal is to extend the expressive power and the modeling comfort of Petri nets. The idea of combining workflow and "nets in nets" is going back to [12], where the authors consider object Petri nets with workflow nets as token nets.

In [24], the authors consider an object-oriented approach in defining synchronization between objects of different classes. For instance, an object might call methods of other objects using the operators: parallel composition, sequential composition and alternative composition. This corresponds to our approach of creating a new net token using these operators.

The idea of controlled modification of token nets is also considered for high-level net and rule (HLNR) systems in [72]. Unlike our approach that easily supports arbitrary (but fixed) nesting level and synchronization between different levels of a nested net, the previous results considered nesting of depth one only. Moreover, [72] carries structural modification of P/T token nets by means of rule tokens, whereas our approach uses predefined and well-known operations, such as sequential and parallel composition.

Our approach is also close to the work of [82] describing dynamic refinement of transitions for super-dual nets which are based on reference nets. As opposed to statical refinement from [64], we have shown that dynamic refinement can make compositional verification more efficient and less expensive. While in Chapter 3, generalized soundness was necessary to prove soundness in a compositional way in a static setting, we have given here a new formalism which allows "dynamic compositional" verification of soundness requiring 1-soundness of net tokens. The practical application in the medical domain of our approach has been reported in [65].

4.6 Conclusion

In this chapter we considered two correctness properties for adaptive nets: soundness and circumspectness. Soundness is in principle a modification of the definition for classical workflow nets. Circumspectness is a new notion induced by the need to verify the enabledness of exception handling and proper decision making. We show how to verify soundness and circumspectness in a compositional manner by reducing it to the verification of corresponding properties of a finite abstraction of adaptive workflow nets.

Using Algorithms 4 and 5 would allow building *correct and robust by construction adaptive workflow*. Like in declarative workflow languages [10], at

certain points in the execution, it is allowed to specify adaptations of the behavior according to new protocols that can in principle be provided at runtime. The advantage of adaptive nets is that one is able to reason about the changed behavior as a whole and the complexity is lower since every protocol is verified separately.

5

Verifying Extended Event-driven Process Chains

Business processes are becoming more and more complex and at the same time their correctness is becoming a critical issue: The costs of errors in business information systems are growing due to the growing scale of their application and the growing degree of automation. In this chapter we consider extended Event-driven Process Chains (eEPCs), a language which is widely used for modeling business processes, documenting industrial reference models and designing workflows. We describe how to translate eEPCs into timed colored Petri nets in order to verify processes given by eEPCs using CPN Tools.

The chapter is based on [61].

5.1 Introduction

Event-driven Process Chains (EPC) [77, 123] is a popular language for modeling business processes, documenting industrial reference models and designing workflows. EPCs describe the flow of control of business processes as a chain of functions, events, and logical connectors. Functions represent activities in a business process. An event expresses a precondition (trigger) for a function or a postcondition that signals the termination of a function. Logical connectors **and**, **or**, and **xor** are used according to their names to build the control flow of a process in a natural way.

EPCs extended with data, resources, time and probabilities, called *extended EPCs (eEPCs)* [123], are intensively used in commercial tools like Architecture of Integrated Information Systems (ARIS) [77] and SAP R/3 [76]. These tools support modeling and simulation of organizational processes with eEPCs, and they are widely used in such branches of industry and consultancy as banks, insurance companies, transportation. The complexity of business processes in these branches is growing throughout the years. Due to informatisation, which concerns all aspects of organizational activities, less and less manual work is involved into the supervision and execution of business processes. This accelerates processes significantly, but also puts higher requirements to the correctness of process specifications, since an error in a process design could demonstrate itself in an automated system too late, when it might already cause a snowball effect.

EEPCs are intensively used in practice although there is no formal definition and semantics provided by ARIS ToolSet. The tool only provides an operational semantics in terms of their simulator. We choose to use an available tool for modeling, simulation and analysis of the constructed model, e.g. model checking which covers the whole system behavior, and we provide a translation from eEPCs to the input language of this tool. Petri nets are appropriate for modeling EPCs since all EPC elements can be translated to places and transitions of Petri nets in a natural way (see e.g. [2, 43, 86]). Extended EPCs have such additional features as data, time and probabilities. Therefore timed colored Petri nets (TCPNs) [74] are a natural choice for modeling eEPCs. CPN Tools [75, 37] provides modeling, simulation, and model checking options for TCPNs and thus satisfies our requirements.

In this chapter, we provide a formal definition of eEPCs and present their semantics in terms of a transition system. We provide a translation from eEPCs to TCPNs and describe how we can analyze the behavior of an eEPC with CPN Tools. Furthermore, we show that the translation preserves the eEPC behavior up to branching bisimulation. We conclude by comparing our method to other approaches for formalizing the syntax and the semantics of EPCs.

The rest of the chapter is organized as follows. In Section 5.2 we describe the syntax of eEPCs. In Section 5.3 we provide the semantics of eEPCs as used in practice. Section 5.4 gives a translation from eEPCs to timed colored Petri

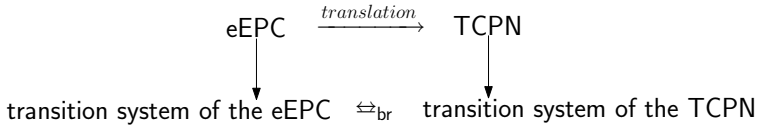


Fig. 18: Overview of this chapter

nets and discusses some verification issues. In Section 5.5 we give an account of related work and discuss some direction for future work.

5.2 Syntax of Extended Event-driven Process Chains

In this section, we give a brief description of the syntax of eEPCs taking into account requirements given in [77] as well as the ones imposed by practice. We use ARIS [41, 73, 77, 123] as a reference point of our study. Our approach can be applied to other tools supporting eEPCs as well, since they are based on the same concepts.

ARIS offers a conceptual framework for describing companies, their organizational structure, processes and resources (material as well as human). In addition to process modeling, ARIS offers the possibility to analyze process performance based on simulation. In order to structure process modeling and to show different angles of an organization, ARIS distinguishes five main views:

Data view uses the entity-relationship models (ERM) to design data models: entities (e.g. data objects of the environment that are processed by the system), their attributes and relationships between entities;

Function view describes functions as tasks performed on objects to support different company goals; it includes descriptions of procedures, processes, subfunctions and elementary functions;

Organization view models the relations between company units and the classification of these units in the organizational hierarchy;

Product/service view describes the products and services produced by the company as a result of human act or technical procedures;

Control view integrates the previously mentioned views and defines the dynamic, behavioral aspects. The control flow of a process is described with an EPC extended with the description of the resources and data involved in the process, and timed and probabilistic aspects of the behavior.

The control view is essential for process verification, so we concentrate our study on this view. In the sequel, we define generic EPCs and extend them to eEPCs as they are presented in the control view of ARIS.

5.2.1 Syntax of EPCs

First, we give a definition of a generic EPC.

Definition 5.1. [EPCs]
An event-driven process chain (EPC) *is a weakly connected directed graph* $G = (V, A)$ *that satisfies the following properties:*

1. *The set* V *is the union of three pairwise disjoint sets* E, F *and* C, *where*
   - E *is the set of* events. $E = E_s \cup E_f \cup E_i$, *where* E_s, E_f *and* E_i *are pairwise disjoint sets of* start events, final events, *and* internal events *respectively, with* $|E_s| \geq 1$ *and* $|E_f| \geq 1$;
   - $F \neq \emptyset$ *is a set of* functions;
   - C *is a set of* connectors *of types* **xor, or, and**, *i.e.* $C = C_{\mathbf{xor}} \cup C_{\mathbf{or}} \cup C_{\mathbf{and}}$, *where* $C_{\mathbf{xor}}$, $C_{\mathbf{or}}$ *and* $C_{\mathbf{and}}$ *are pairwise disjoint sets. Furthermore, each of these sets is partitioned into two pairwise disjoint sets representing* split *and* join *connectors:* $C_{\mathbf{xor}} = C_{\mathbf{xs}} \cup C_{\mathbf{xj}}$, $C_{\mathbf{or}} = C_{\mathbf{os}} \cup C_{\mathbf{oj}}$ *and* $C_{\mathbf{and}} = C_{\mathbf{as}} \cup C_{\mathbf{aj}}$, *and* $C_{\mathbf{s}}$ *stands for set of split connectors, and* $C_{\mathbf{j}}$ *stands for the set of join connectors;*
2. *Every element from the set* A *of arcs connects two different vertices. Moreover,*
   - ${}^\bullet e_s = \emptyset$ *and* $e_f^\bullet = \emptyset$, *for each* $e_s \in E_s$ *and* $e_f \in E_f$;
   - $|n^\bullet| = 1$ *for each* $n \in F \cup E_i \cup E_s$, *and* $|{}^\bullet n| = 1$ *for each* $n \in F \cup E_i \cup E_f$;
   - *each split connector* $c \in C_{\mathbf{s}}$ *satisfies* $|{}^\bullet c| = 1$ *and* $|c^\bullet| > 1$; *similarly each join connector* $c \in C_{\mathbf{j}}$ *satisfies* $|{}^\bullet c| > 1$ *and* $|c^\bullet| = 1$;
3. *Each vertex is on a path from a start event to a final event, i.e. for any* $n \in V$, *there is a path* σ *from some* $e_s \in E_s$ *to some* $e_f \in E_f$, *such that* $n \in \sigma$;
4. *Functions and events alternate along the control flow, i.e. each path starting in an event* $e \in E \setminus E_f$ *and ending in an event* $e_f \in E_f$ *has a prefix of the form* $e\sigma_c f$, *where* $\sigma_c \in C^*$ *and* $f \in F$. *Similarly, for each path* σ *starting in a function* $f \in F$ *and ending in an event* $e_f \in E_f$ *there is a prefix* $f\sigma_c e$, *where* $\sigma_c \in C^*$ *and* $e \in E \setminus E_s$;
5. *Events do not precede the* **xor** *and the* **or** *split, i.e.* $\forall c \in C_{\mathbf{xs}} \cup C_{\mathbf{os}} : {}^\bullet c \cap E = \emptyset$;
6. *There is no cycle that consists of connectors only, i.e. for any path* $\sigma = v_0 v_1 \ldots v_n \in C^+ : n \geq 2 : v_0 \neq v_n$.

5.2.2 Syntax of Extended EPCs

The control view of an eEPC has an EPC as a skeleton. Data attributes, resources, time and probabilities are linked to different EPC elements to form an extended EPC.

Functions represent activities that may take time, may require access to diverse resources and may perform operations on some data or resources.

Functions that perform operations on data attributes are annotated with expressions denoting the operation performed (see for example Figure 19(a)). Personnel, material or information resources can be used to execute functions.

We call these objects capacity resources, since they are characterized by minimum and maximum capacities to run the process which are delimiting the capacity resource domain. Functions are annotated with a nonnegative integer or real constant denoting the number of resources required, produced or consumed. In Figure 19(b), function *finish products* produces 1000 items of resource $r1$ with startup capacity 1000 and capacity domain $[r1_{\min}, r1_{\max}]$, where $r1_{\min}$ = 100 is the lower capacity bound and $r1_{\max}$ = 5000 is the upper capacity bound. Moreover, the function consumes 500 items of resource $r2$ with the same capacity domain and sturtup capacity.

Furthermore, functions can be either *timed*, i.e. they may have a duration, or *immediate*, i.e they take zero time. The duration of each timed function is described by a probability distribution.

Events define either conditions on data attributes or resource capacities, or triggers from elements outside the process.

Processes are instantiated at **start event sets**. Each such start event set[4] contains *start events* which are synchronized, i.e. a process is started at the same time at all the events of the respective set. A probability distribution is assigned to each start event set, denoting the frequency with which process instances are created for the events in the respective set.

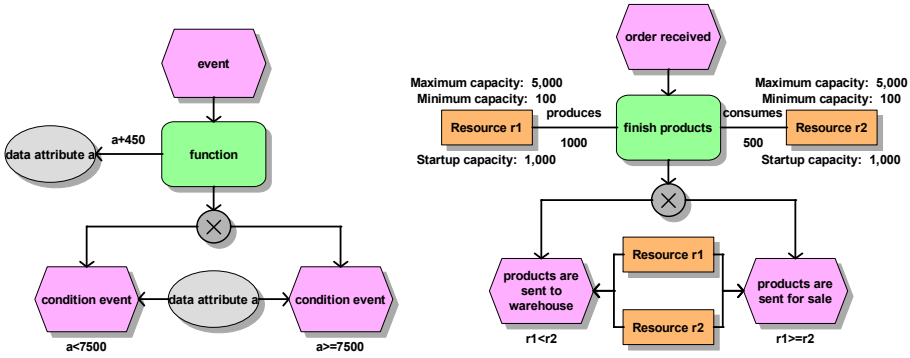
End events are events with no outgoing arc, at which process instances are stopped.

Conditions (boolean expressions) on data attributes or on resources determine the terms of the respective event. An event that follows an **or** split or an **xor** split connector and is determined by a condition is called a **condition event**. Condition events may have **attributes** or **capacity resources** connected to them and conditions are specified as:

- conditions on one operand that have a constant value of the same type as the attribute or the capacity value of a resource as comparison criterion. Figure 19(a) shows two condition events annotated with boolean expressions on a data attribute;
- conditions on two operands that compare two attribute values or the capacity values of two resources. In Figure 19(b), the condition events *products are sent to warehouse* and *products are sent for sale* are annotated with the boolean expressions $r1 < r2$ and $r1 \geq r2$, where $r1$ and $r2$ are the variables corresponding to resources *Item 1* and *Item 2*, respectively.

The rest of events are used to model triggers from outside the process and they have a probability value assigned. This value is used during the simulation to determine whether the execution stops or continues at the respective event. Probability values for events following **and split** connectors are 1 since the execution cannot stop at events following such a connector. The sum of probability values for events following **xor split** connectors is exactly 1 as the execution can continue only on one outgoing branch. Furthermore, the sum of

[4] In ARIS, event diagrams are used to represent start event sets.



(a) Condition events on a data attribute

(b) Condition events on resources

Fig. 19: Condition events on data attributes and resources

probability values for events following **or split** connectors is greater than 1 as the execution can continue on one or more outgoing branches.

Or join connectors may also contain some timeout information, called *synchronization timeout*. The synchronization timeout represents the amount of time with which the firing of the **or join** connector for one process instance is delayed after a first representative of the process instance arrives at the connector while waiting for other representatives of the process instance to arrive at the **or join** connector.

We give a formal definition of eEPCs as they are used in ARIS Toolset.

Definition 5.2. [EEPC]
An extended event-driven process chain (eEPC) is defined by the tuple $G_e = \langle G, \mathcal{A}, \mathcal{R}, Type, Expr, PDF, \mathbf{Pr} \rangle$*, where*

- G *is an underlying EPC.*
- $\mathcal{A}$ *is a set of data attributes. We decompose* $\mathcal{A}$ *into a family* $\bigcup_{f \in F} \mathcal{A}_f$ *of sets of data attributes indexed by the function performing operations on them.*
- $\mathcal{R}$ *is a set of capacity resources. We decompose* $\mathcal{R}$ *into a family* $\bigcup_{f \in F} \mathcal{R}_f$ *of sets of resources indexed by the function performing operations on them. Note that these sets are not disjoint as several functions can perform operations on the same resource. Moreover, we consider a partition of the set of resources used by a function* f *into used, produced and consumed resources, i.e.* $\mathcal{R}_f = \mathcal{R}_f^u \cup \mathcal{R}_f^p \cup \mathcal{R}_f^c$*. Note that a function can perform just one type of operation on a resource.*
- $Type$ *maps each attribute to one of the types* **Text, Enum**, $\mathbb{B}$, $\mathbb{Z}$, *or* $\mathbb{R}$ *and each capacity resource* $r \in \mathcal{R}$ *to a real or integer subtype* $[r_{\min}, r_{\max}]$*, where* $r_{\min}$ *and* $r_{\max}$ *are the minimum and maximum capacities of resource* r*.*
- $Expr = Expr^b \cup \bigcup_{x \in \mathcal{R} \cup \mathcal{A}} Expr_x$ *maps condition events and functions to expressions on the attributes or capacity resources linked to them as follows:*

  - $E_c \subseteq E \cap (C_{\mathbf{xs}} \cup C_{\mathbf{os}})^\bullet$ *denotes the set of* **condition events**, *i.e. events following* **or split** *and* **xor split** *connectors that have conditions on data attributes or resources. Every condition event* $e \in E_c$ *is mapped to a boolean expression* $Expr^b(e)$ *of the form* v_1 **x** v_2 *or* v_1 **x** c, *where* v_1, v_2 *are either attributes or resource capacities,* c *is a constant so that* $Type(v_1) = Type(v_2) = Type(c)$ *and* **x** *is the comparison operator compatible with the types used.*
  - *every function* f *performing an operation on an attribute* $\mathfrak{a}$ *is mapped to an expression* $Expr_{\mathfrak{a}}(f)$, *having the form* $\mathfrak{a} := c$, *where* c *is a constant value with* $Type(\mathfrak{a}) = Type(c)$, *or* $\mathfrak{a} := \mathfrak{a}$ **x** c, *with* $Type(\mathfrak{a}) = \mathbb{Z}$ *or* $Type(\mathfrak{a}) = \mathbb{R}$, *constant* c *with* $Type(\mathfrak{a}) = Type(c)$ *and* $\mathbf{x} \in \{+, -, *\}$.
  - $Expr_r(f)$ *maps function* f *using, producing or consuming a resource* $r \in \mathcal{R}_f$ *to constant* $c_r^f \in Type(r)$, *denoting the quantity of resources used, consumed or produced.*
- $F_t \subseteq F$ *denotes the* timed *functions, and* $C_t \subseteq C_{\mathbf{oj}}$ *denotes the set of* **or join** *connectors with synchronization timeout. We consider the set of start events* E_s *partitioned into start event sets, i.e.* $E_s = \bigcup_{d \in I_s} E_s^d$, *for some index set* I_s.
  $PDF = \bigcup_{k \in (F_t \cup C_t \cup I_s)} pdf_k$ *denotes a family of continuous or discrete probability distributions*[5] *for the duration of timed functions, for the synchronization timeouts of timed* **or join** *connectors, and for the delays of start event sets.*
- $\mathbf{Pr}: E \setminus E_s \setminus E_c \to [0,1]$ *which maps events to their probability values such that:*
  - $\sum_{e \in E \cap c^\bullet} \mathbf{Pr}(e) = 1$, *for each* **xor split** *connector* $c \in C_{\mathbf{xs}}$ *with* $E \cap c^\bullet \neq \emptyset$;
  - $\sum_{e \in E \cap c^\bullet} \mathbf{Pr}(e) \geq 1$, *for each* **xor split** *connector* $c \in C_{\mathbf{os}}$ *with* $E \cap c^\bullet \neq \emptyset$;
  - $\mathbf{Pr}(e) = 1$ *for each* $e \in C_{\mathbf{as}}{}^\bullet \cap E$.

Figure 20 shows an eEPC modeling a part of a trade matching process taking place in a company. The process checks the timely receipt of a confirmation, administrates the trade internally and matches the confirmation against the internal data before the release process can be started.

The process starts with the event *trade executed (deal made)* that triggers the function *monitor receipt of trade confirmation and administration.* This results in a change of the data attribute *trade use.* The execution is then split into two parallel threads, which is modeled by an **and split** . The left thread models the check whether the confirmation of the trade has been received electronically or manually by means of an **xor split** and two condition events: *Received trade confirmation (manual)* and *Received trade confirmation (electronic)* that are linked to a data attribute *Trade use.* The second thread models the check whether the trade is administered for manual or electronic use by means of conditions on the attribute *trade use.*

[5] A probability distribution $pdf \in PDF$ refers here to a probability distribution function (we do not mention the word function in order to avoid confusion with functions as nodes of eEPCs).

The two **and join** connectors make sure that the matching process continues either manually or electronically. The visual (manual) check is performed by the function *visual trade check* and results in the event *trade checked*. The result of the electronic matching of the internal information with BLIM messages has 40% probability to succeed. In case the trade has been matched either manually or electronically, which is modeled by an **xor join** , the process is released by *start release process*. If the data registered internally does not match the information contained in the BLIM message, the message is rejected.

5.3 Semantics of eEPCs

We introduce first the notions of a *process folder* and a *state* of an eEPC necessary to define the semantics of eEPCs.

A **process folder** is an object that resides at a node (function or start event) or at an arc. Furthermore, it carries a folder number and a time stamp denoting the value of the timer associated to the folder. Time stamps are either nonnegative numbers indicating the delay after which the folder may be used, or $\perp$, denoting that the timer of the process folder is off and the folder can be used directly. A **state** of an eEPC is defined by a multiset of process folders together with a valuation of data attributes and capacity resources.

For the rest of the chapter, we assume the time domain $\mathbb{N}_\perp = \mathbb{N} \cup \{\perp\}$ and probability distributions for durations to be discrete. We denote the domain of a discrete probability distribution $pdf \in PDF$ by $Dom(pdf) \subseteq \mathbb{N}$. The same approach can be applied for continuous time and continuous probability distributions. We consider process folder numbers to take values from $\mathbb{N}$. Formally:

Definition 5.3. [PROCESS FOLDER, STATE]
Let $G_e = \langle G, \mathcal{A}, \mathcal{R}, Type, Expr, PDF, \mathbf{Pr}\rangle$ *be an eEPC. A* process folder *is a tuple* $\mathfrak{f} = (n, (i, \iota))$ *where* $n \in E_s \cup F \cup A$ *is a start event, a function or an arc,* $i \in \mathbb{N}$ *is a process folder number and* $\iota \in \mathbb{N}_\perp$ *is a time stamp. A* state *of* G_e *is a tuple* $s = (m, Val)$*, where* m *is a multiset of process folders, i.e.* $m\colon ((F \cup A \cup E_s) \times (\mathbb{N} \times \mathbb{N}_\perp)) \to \mathbb{N}$*, and* Val *is a valuation function that maps every resource* $r \in \mathcal{R}$ *into some value* $Val(r) \in Type(r)$ *and every data attribute* $\mathfrak{a} \in \mathcal{A}$ *to some value* $Val(\mathfrak{a}) \in Type(\mathfrak{a})$*.*

We denote the time stamp of a process folder $\mathfrak{f}$ by $\mathfrak{f}_t$. We will say that a process folder has its timer *off* when $\mathfrak{f}_t = \perp$ and its timer *on* when $\mathfrak{f}_t \geq 0$. We call a process folder with the timer on *active* if it has the time stamp 0. If $\mathfrak{f}_t > 0$, the process folder is *waiting* to become active.

Every start event of a start event set has initially an active process folder with the index of the start event set as its folder number. The initial state is thus $s_0 = (m_0, Val_0)$ and contains one active process folder on every start event, i.e $m_0 = \sum_{d \in I_s} \sum_{e_s \in E_s^d} [(e_s, (i, 0))]$ and Val_0 is the initial valuation of resources and data attributes, i.e. $Val_0\colon \mathcal{R} \cup \mathcal{A} \to \bigcup_{x \in \mathcal{R} \cup \mathcal{A}} Type(x)$ so that for

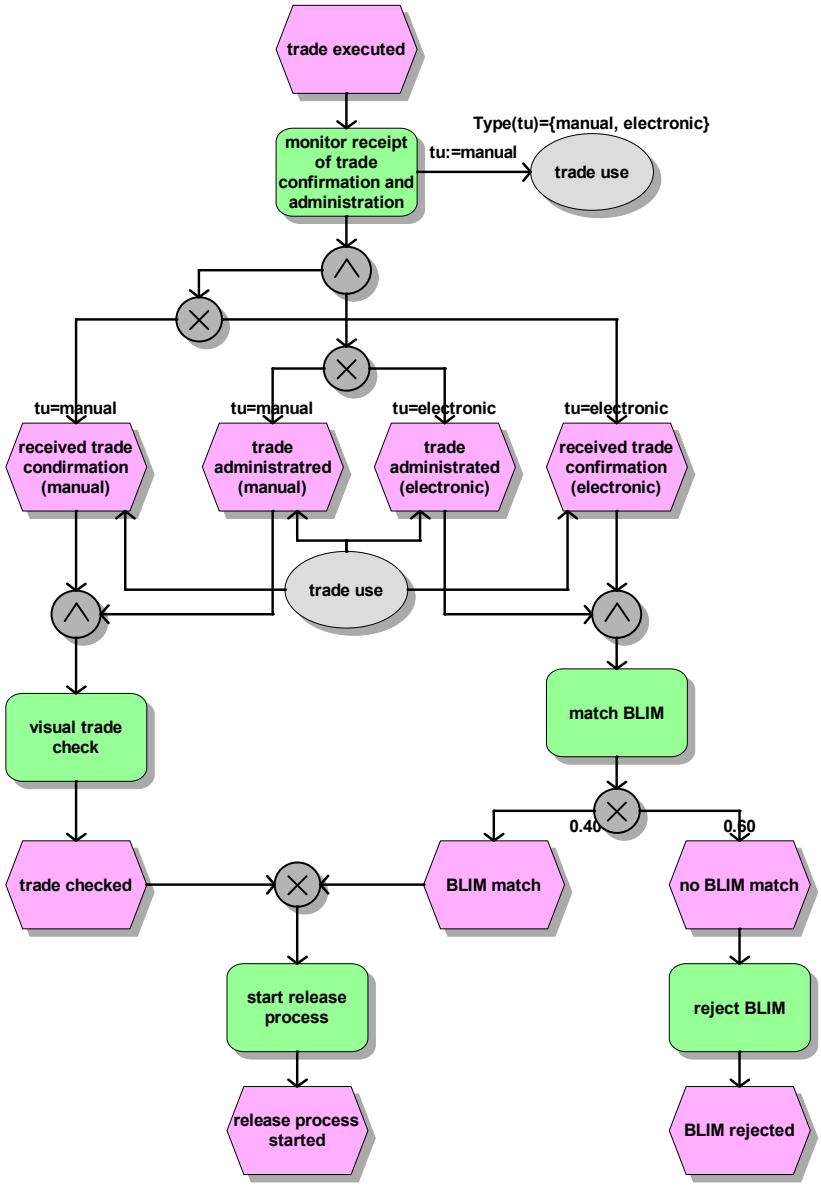


Fig. 20: Trade matching eEPC

each $r \in \mathcal{R}$, $Val_0(r)$ is the startup capacity of r and for each attribute $a \in \mathcal{A}$, $Val_0(a) \in Type(a)$ is the initial value of a.

Probability distributions are used in eEPCs to model the behavior of the environment or to describe nondeterminism in the system (when decisions need to be made), and for performance analysis. Since all events $e \in E$ that have $\mathbf{Pr}(e) > 0$ can occur and the errors in a model (eEPC) having probabilistic events can thus be detected on the model without probabilities, we do not take probabilities further into consideration, as they are irrelevant to our verification purposes. In order to express nondeterminism without probabilities, we extend the mapping $Expr^b$ to non-condition events and set $Expr^b(e) = true$, for every event $e \in (E \setminus E_c) \cap (C_{\mathbf{os}} \cup C_{\mathbf{xs}})^\bullet$, and subsequently extend the set of *condition events* to all events following an **or split** and **xor split** connector, i.e. $E_c = (C_{\mathbf{os}} \cup C_{\mathbf{xs}})^\bullet \cap E$. We denote the set of variables of an expression $exp \in \bigcup_{e \in E_c} Expr^b(e)$ by $Var(exp)$.

Let *eval* be the *evaluation function* for expressions, such that:

- $eval(Expr^b(e), Val) \in \mathbb{B}$ is the evaluation function of the boolean expression of condition events $e \in E_c$ in the valuation Val.
- $eval(Expr_{\mathfrak{a}}(f), Val) \in Type(\mathfrak{a})$ is the evaluation function of $Expr_{\mathfrak{a}}(f)$ on some data attribute $\mathfrak{a} \in \mathcal{A}_f$ in the valuation Val that computes a new value for $\mathfrak{a}$ from the right hand side expression of $Expr_{\mathfrak{a}}(f)$.

A resource can be used, produced or consumed by several functions at the same time. Each such operation (function use, produce, consume) has two separate steps:

1. acquirement of the resource: at this stage the resource capacity is checked for availability (for consumption, use or production) and the consumption and acquirement part of the use operations are executed;
2. release of the resource: at this stage the production and the release part of the use operations are executed.

The availability check for a resource that is going to be used is limited to verifying whether a certain amount of resource items can be claimed; however, it is still possible for the resource to reach its upper bound capacity by production done by some other function(s), which can makes the release of the used items increase the value of the respective resource over its upper bound. For any $r \in \mathcal{R}$, we introduce a variable $\bar{r}$ such that $Val_0(r) = Val_0(\bar{r})$ in order keep track of values of the resources that are going to be changed before the change actually happens. We denote the set of variables newly introduced by $\bar{\mathcal{R}}$.

Let $\Sigma = I_s \cup (E \setminus (E_s \cup E_c \cup E_f)) \cup \{f_a, f_r | f \in F\} \cup C_{\mathbf{xor}} \cup (C_{\mathbf{or}} \setminus C_t) \cup \{c_w, c_f | c \in C_t\} \cup C_{\mathbf{and}} \cup \mathbb{N}^+$. We describe the semantics of an eEPC by means of a transition relation between states as follows:

Definition 5.4. [Semantics of eEPCs]
Let $G_e = \langle G, \mathcal{A}, \mathcal{R}, Type, Expr, PDF, \mathbf{Pr} \rangle$ be an eEPC. The semantics of an eEPC are given by a labeled transition system $U = \langle \mathbb{S}, \Sigma, \rightarrow, s_0 \rangle$, where $\mathbb{S}$ is the

set of states of G_e, Σ is the set of elements involved in the application of the rules, s_0 is the initial state, $\rightarrow \subseteq \mathbb{S} \times \Sigma \times \mathbb{S}$ is a transition relation described by the rules (a) – (k) below. Let $s = (m, \mathrm{Val})$ and $s' = (m', \mathrm{Val}')$ be two states in $\mathbb{S}$.

(a) start event set rule *Let E_s^d be a start event set such that there is a process folder on each of its start events and all the folders have the same folder number and are active. Then a step can be taken that results in removing all the folders on the events of E_s^d and producing a process folder on each of the outgoing arcs of E_s^d which have the same folder number as the original process folders and their timers are set to off. Furthermore, a new process folder is generated on each event of E_s^d; all these new process folders have the same newly generated folder number and the same time stamp which is drawn from the probability distribution of the start event set.*

Formally, if $(e, (i, 0)) \in m$ for all $e \in E_s^d$, $d \in I_s$ and $i \in \mathbb{N}$, then $(m, \mathrm{Val}) \xrightarrow{d} (m', \mathrm{Val})$, with

$$m' = m - \sum_{e \in E_s^d} [(e, (i, 0))] + \sum_{e \in E_s^d} [(a_e^{out}, (i, \perp))] + \sum_{e \in E_s^d} [(e, (i + |I_s|, \iota)],$$

for some $\iota \in Dom(pdf_d)$.

(b) event rule *Let a_e^{in} be the incoming arc for an event e such that there is a process folder on a_e^{in}. Then, the process folder can be removed from a_e^{in} and placed on the outgoing arc of the event a_e^{out}. Formally, if $(a_e^{in}, (i, \perp)) \in m$ for some $i \in \mathbb{N}$ and $e \in E \setminus (E_s \cup E_c \cup E_f)$, then $(m, \mathrm{Val}) \xrightarrow{e} (m', \mathrm{Val})$ with $m' = m + [(a_e^{out}, (i, \perp))] - [(a_e^{in}, (i, \perp))]$.*

(c) function rule *Let a_f^{in} be the incoming arc of a function f such that there is a process folder on a_f^{in}. The function rule consists of two steps:*

function acquiring *if the resources may be used, consumed or produced, the process folder is removed from the incoming arc of the function, and a new process folder having the same folder number as the original folder and a time stamp from the time distribution interval of the function is placed on the function, and resources are consumed. Formally, if*

- $(a_f^{in}, (i, \perp)) \in m$, *for some $f \in F$ and $i \in \mathbb{N}$,*
- $\mathrm{Val}(r) - \mathrm{Expr}_r(f) \geq r_{\min}$ *for all $r \in \mathcal{R}_f^u \cup \mathcal{R}_f^c$,*
- $\mathrm{Val}(\bar{r}) + \mathrm{Expr}_r(f) \leq r_{\max}$ *for all $\bar{r} \in \bar{\mathcal{R}}_f^p$,*

then $(m, \mathrm{Val}) \xrightarrow{f_a} (m', \mathrm{Val}')$, where $m' = m - [(a_f^{in}, (i, \perp))] + [(f, (i, \iota))]$ and

- $\iota = 0$ *if $f \in F \setminus F_t$ and $\iota \in Dom(pdf_f)$ if $f \in F_t$,*
- $\mathrm{Val}'(r) = \mathrm{Val}(r) - \mathrm{Expr}_r(f)$ *for all $r \in \mathcal{R}_f^c \cup \mathcal{R}_f^u$,*
- $\mathrm{Val}'(\bar{r}) = \mathrm{Val}(\bar{r}) - \mathrm{Expr}_r(f)$ *for all $\bar{r} \in \bar{\mathcal{R}}_f^c$,*
- $\mathrm{Val}'(\bar{r}) = \mathrm{Val}(\bar{r}) + \mathrm{Expr}_r(f)$ *for all $\bar{r} \in \bar{\mathcal{R}}_f^p$ and*

– $Val'(x) = Val(x)$ *for all* $x \in (\mathcal{R} \cup \bar{\mathcal{R}} \cup \mathcal{A}) \setminus (\mathcal{R}_f^c \cup \mathcal{R}_f^u \cup \bar{\mathcal{R}}_f^c \cup \bar{\mathcal{R}}_f^p)$.

function release *Let* f *be a function such that there is an active process folder on it. Then, the active process folder is removed from the function, and a new process folder is produced on the outgoing arc of the function; this new folder has the same folder number as the removed one and the timer off; attributes are evaluated and resources are produced or released.*

Formally, if $(f,(i,0)) \in m$, *for some* $f \in F$ *and* $i \in \mathbb{N}$, *then* $(m, Val) \xrightarrow{f_r} (m', Val')$, *with* $m' = m - [(f,(i,0))] + [(a_f^{out},(i,\perp))]$, *where* $Val'(\mathfrak{a}) = eval(Expr_{\mathfrak{a}}(f), Val)$ *for all* $\mathfrak{a} \in \mathcal{A}_f$, $Val'(r) = Val(r) + Expr_r(f)$ *for all resources* $r \in \mathcal{R}_f^p \cup \mathcal{R}_f^u$ *produced or used, and* $Val'(x) = Val(x)$ *for all* $x \in (\mathcal{R} \cup \bar{\mathcal{R}} \cup \mathcal{A}) \setminus (\mathcal{A}_f \cup \mathcal{R}_f^p \cup \mathcal{R}_f^u)$.

(d) and split rule *Let* a_c^{in} *be the incoming arc of an* **and split** *connector* c *such that there is a process folder on* a_c^{in}. *The rule results in removing the process folder from* a_c^{in} *and placing a process folder on each outgoing arc of the* **and split** *connector such that all new process folders have the same folder number as the removed folder. Formally, if* $(a_c^{in},(i,\perp)) \in m$, *for some* $i \in \mathbb{N}$ *and* $c \in C_{\mathbf{as}}$, *then* $(m, Val) \xrightarrow{c} (m', Val)$ *and*

$$m' = m + \sum_{a' \in A_c^{out}} [(a',(i,\perp))] - [(a_c^{in},(i,\perp))].$$

(e) xor split rule *Let* a_c^{in} *be the incoming arc of an* **xor split** *connector* c *such that there is a process folder on* a_c^{in}. *Then the process folder is removed from the arc* a_c^{in} *and*

– *if the* **xor split** *leads to a non-final condition event whose boolean expression is evaluated to true, a process folder with the same number as the removed one is placed on the outgoing arc of the event;*
– *if there is an outgoing arc of the* **xor split** *leading to a final condition event whose boolean expression is evaluated to true or to another connector, then a process folder with the same number as the removed one is placed on the respective arc.*

Formally, if $(a_c^{in},(i,\perp)) \in m$, *for some* $i \in \mathbb{N}$ *and* $c \in C_{\mathbf{xs}}$, *then* $(m, Val) \xrightarrow{c} (m', Val)$ *and* $m' = m + [(a',(i,\perp))] - [(a_c^{in},(i,\perp))]$, *where*

– $a' = a_e^{out}$ *if* $eval(Expr^b(e), Val) = true$ *for some event* $e \in (E \setminus E_f) \cap c^\bullet$, *or*
– $a' = (c,e)$, *if* $eval(Expr^b(e), Val) = true$ *for some final event* $e \in c^\bullet \cap E_f$, *or*
– $a' = (c,c')$, *for some* $c' \in C \cap c^\bullet$.

(f) or split rule *Let* a_c^{in} *be the incoming arc of an* **or split** *connector* c *such that there is a process folder on* a_c^{in}. *The rule results in removing this folder from* a_c^{in} *and placing a process folder with the same folder number on at least one of the outgoing arcs of the non-final condition events that have a true boolean expression or the outgoing arcs of the* **or split** *connector*

leading to final condition events with boolean expression evaluated to true or to connectors.
Formally, if $(a_c^{in}, (i, \perp)) \in m$, *for some* $i \in \mathbb{N}$ *and* $c \in C_{\mathbf{os}}$, *then* $(m, Val) \xrightarrow{c} (m', Val)$ *and* $m' = m - [(a_c^{in}, (i, \perp))] + \sum_{a \in A' \cup A''} [(a, (i, \perp))]$, *where* $A' \subseteq \{a_e^{out} | e \in c^{\bullet} \cap (E_c \setminus E_f) \wedge eval(Expr^b(e), Val) = true\}$ *and* $A'' \subseteq \{(c, e) \in A_c^{out} | e \in c^{\bullet} \cap (E_c \cap E_f) \wedge eval(Expr^b(e), Val) = true\} \cup \{(c, c') \in A_c^{out} | c' \in C\}$ *and* $A \cap A'' \neq \varnothing$.

(g) and join rule *Let* A_c^{in} *be the set of all incoming arcs of an* **and join** *connector* c. *If all arcs of* A_c^{in} *have process folders with the same folder number, the rule can be applied resulting in the removal of these process folders from* A_c^{in}, *and the production of a process folder with the same process folder number as the original folders on the outgoing arc of the connector.*
Formally, if there is a $c \in C_{\mathbf{aj}}$ *such that* $(a, (i, \perp)) \in m$, *for all arcs* $a \in A_c^{in}$ *and some* $i \in \mathbb{N}$, *then* $(m, Val) \xrightarrow{c} (m', Val)$ *and* $m' = m + [(a_c^{out}, (i, \perp))] - \sum_{a \in A_c^{in}} [(a, (i, \perp))]$.

(h) xor join rule *Let* a *be an incoming arc of an* **xor join** *connector such that there is a process folder on* a. *Then, the folder is removed from* a *and placed on the outgoing arc of the connector.*
Formally, if $(a, (i, \perp)) \in m$ *for some* $i \in \mathbb{N}$, $c \in C_{\mathbf{xj}}$ *and* $a \in A_c^{in}$, *then* $(m, Val) \xrightarrow{a} (m', Val)$ *and* $m' = m + [(a_c^{out}, (i, \perp))] - [(a_c^{in}, (i, \perp))]$.

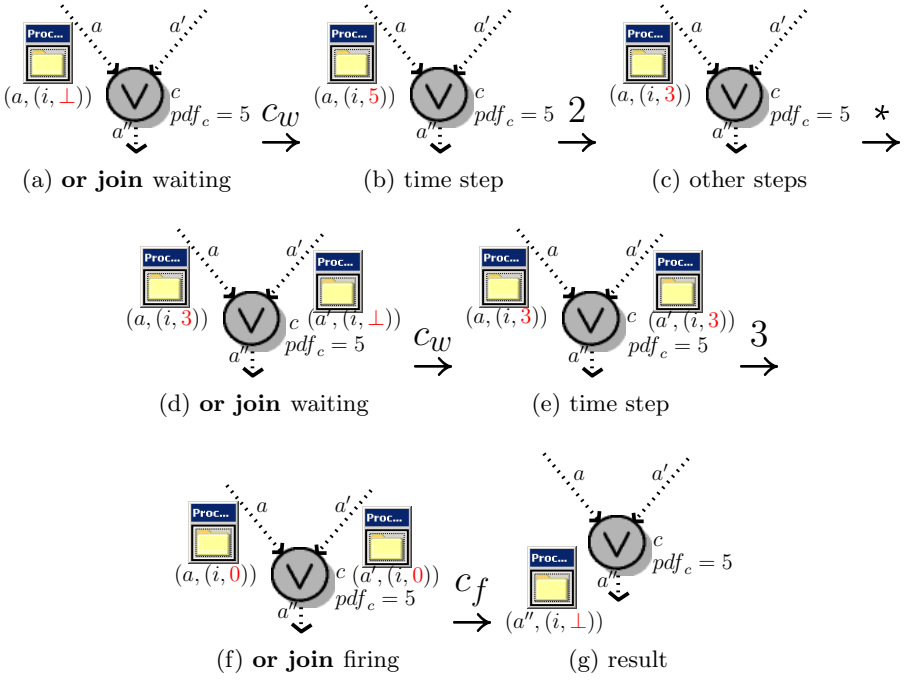
(i) or join waiting rule *Let* $\mathfrak{F}_{c,i} \neq \varnothing$ *be the bag of process folders on the incoming arcs of an* **or join** *connector that have process folders with timers off and the same folder number* $i \in \mathbb{N}$. *Let* $\mathfrak{F}'_{c,i}$ *the bag of waiting process folders present on the incoming arcs of some* **or join** *connector* c *having the same process number* i *with time stamp* $\iota > 0$. *The rule results in modifying the time stamp of the process folders from* $\mathfrak{F}_{c,i}$ *such that they have as a time stamp the synchronization timeout in case there are no waiting process folders (*$\mathfrak{F}'_{c,i} = \varnothing$*) or the time stamp of the waiting process folders otherwise (*$\mathfrak{F}'_{c,i} \neq \varnothing$*).*
Let $\mathfrak{F}_{c,i} = \sum_{\mathfrak{f}=(a,(i,\perp)) \in m \wedge a \in A_c^{in}} m(\mathfrak{f})[\mathfrak{f}]$ *and* $\mathfrak{F}'_{c,i} = \sum_{\mathfrak{f}=(a,(i,\iota)) \in m \wedge a \in A_c^{in}} m(\mathfrak{f})[\mathfrak{f}]$, *for some* $c \in C_t$, $\iota > 0$ *and* $i \in \mathbb{N}$ *so that* $\mathfrak{F}_{c,i} \neq \varnothing$. *If* $\mathfrak{F}'_{c,i} = \varnothing$ *then let* $\iota' = \iota$ *be the time stamp of the process folders in* $\mathfrak{F}'_{c,i}$, *otherwise let* $\iota' \in Dom(pdf_c)$.
Then, $(m', Val') \xrightarrow{c_w} (m', Val)$ *and*

$$m' = m - \mathfrak{F}_{c,i} + \sum_{\mathfrak{f}=(a,(i,\perp)) \in \mathfrak{F}_{c,i}} m(\mathfrak{f})[(a, (i, \iota'))].$$

(j) or join firing rule *Let* $\mathfrak{F}_{c,i}$ *be the non-empty bag of process folders that have non-waiting process folders with the same folder number* $i \in \mathbb{N}$. *Then, the process folders in* $\mathfrak{F}_{c,i}$ *are removed and a new process folder is produced on the outgoing arc of the connector, so that it has the same folder number as the original folders and the timer off.*
Formally, let $\mathfrak{F}_{c,i} = \sum_{\mathfrak{f}=(a,(i,\iota)) \in m \wedge a \in A_c^{in}} m(\mathfrak{f})[\mathfrak{f}]$, *where* $c \in C_{\mathbf{oj}}$, $i \in \mathbb{N}$, $\iota = 0$ *if* $c \in C_t$ *and* $\iota = \perp$ *if* $c \in C_{\mathbf{oj}} \setminus C_t$. *If* $\mathfrak{F}_{c,i} \neq \varnothing$, *then* $(m, Val) \xrightarrow{c_f} (m', Val)$ *and* $m' = m + [(a_c^{out}, (i, \perp))] - \mathfrak{F}_{c,i}$.

(k) time step rule *This rule has the lowest priority, i.e. the rule is applied if no other rule can be applied. Time passage is applied when all process folders with timers on in a state are waiting (have a strictly positive time stamp) and results in decreasing the time stamp of all process folders of the state by the minimal time stamp of the folders with timers on.*
Formally, if $\mathfrak{F}' = \{\mathfrak{f} \in m | \mathfrak{f}_t > 0\} \neq \varnothing$ *and* $\mathfrak{t} = \min\{\mathfrak{f}_t | \mathfrak{f} \in \mathfrak{F}'\} > 0$ *and there is no other state* $s'' \neq s'$ *such that* $s \xrightarrow{x} s''$*, where* $x \in \Sigma \setminus \mathbb{N}^+$*, then* $s \xrightarrow{\mathfrak{t}} s'$*, with* $s' = (m', Val)$*, where* $m' = m + \sum_{(x,(i,\iota))\in\mathfrak{F}'}[(x,(i,\iota-\mathfrak{t}))] - \mathfrak{F}'$.

Example 1. Figure 21 shows an example of a timed **or join** firing when two process folders come at an interval of 2 time units and the timeout for the **or join** connector is 5.



(a) **or join** waiting (b) time step (c) other steps

(d) **or join** waiting (e) time step

(f) **or join** firing (g) result

Fig. 21: Firing sequence

Based on Definition 5.4 we make some remarks.

Remark 1 (Uniqueness of newly generated folder numbers). Folder numbers generated at some start event set E_s^d $(d \in I_s)$ have the form $d + k \cdot n$, where $n = |I_s|$ and $k \in \mathbb{N}$. Therefore, all folder numbers produced at E_s^d are equal

modulo the index number of the start event set, i.e. d. As a result, all process folder numbers that are generated are unique.

Remark 2 (Time progress). The time step rule decreases the time stamp of a waiting process folder until it becomes active (its timer expires). Note that all other steps have the same higher priority than the time progress and their semantics is interleaving. The time progress problem reduces to the situation where no other rule can be applied or to the situation where there is an infinite sequence of (timeless) rules that can be applied. We therefore assume that there is a finite number of process folder numbers such that the states contain a finite number of process folders with the same folder number.

Remark 3 (Resource use, consumption and production). The guards introduced for functions which use, consume and produce resources make the following invariant hold: in every reachable state, the values of both variables for any resources stay within the lower and upper capacity bounds. Before any consumption and production, the possibility of performing such an action is checked on variables from $\mathcal{R}$, and the consumption is performed on both variables from $\mathcal{R}$ and $\bar{\mathcal{R}}$. Before any use of resources, the possibility to acquire the resources is checked, and the respective quantity is consumed. Before any production of resources, the possibility of production of the resource is checked on the variables from $\bar{\mathcal{R}}$, which contains the values of the resources that are processed by functions that are producing resources of that type. The actual values of the resources (given by variables from $\mathcal{R}$) are changed before the resources are released by the functions and thereafter these values is not greater than the upper bound since these condition has been checked previously on the variables from $\bar{\mathcal{R}}$.

Moreover, the value of the variable r and $\bar{r}$ coincide after all the functions which have acquired the resource for use, have released it. This invariant can be easily checked by induction on the number of functions in use of the resource. Hence, after all execution sequences throughout which all the resources that were acquired have been released, the value of the variable r and $\bar{r}$ coincide.

We extend the order relations $\leq$ and $=$ to deal with $\perp$. We define $\perp \leq \iota$ for all $\iota \in \mathbb{N}$. We now state a property related to the time stamps of the process folders on the incoming arcs of an **or join** connector.

Lemma 5.1. *Let $s = (m, Val)$ be a state of an eEPC G_e, let*

$$\mathfrak{F}_{c,i} \stackrel{\text{def}}{=} \sum_{((a_c^{in},i),\iota)\in m} [((a_c^{in}, i), \iota)]$$

be the multiset of process folders with identifier $i \in \mathbb{N}$ on the incoming arcs of a timed **or join** *connector $c \in C_t$ and $((c,i),\iota_1), ((c,i),\iota_2) \in \mathfrak{F}_{c,i}$ be two process folders. Then $\iota_1 \in \mathbb{N}$ and $\iota_2 \in \mathbb{N}$ implies that $\iota_1 = \iota_2$.*

5.4 Verification of eEPCs Using CPN Tools

To verify the correctness of eEPCs, we translate them to a different formalism — (Timed) Colored Petri nets (TCPNs), for which verification tools are available. We use CPN Tools [37] which are based on colored Petri nets [74] as modeling and analysis language. TCPNs combine the expressive power of classical PNs, which are suitable for modeling the logical behavior of a control flow, with the modeling of data and time by means of timed color sets and a global clock. A state of a TCPN is called a *marking* and represents the distribution of (timed) colored tokens on places. Tokens belonging to a timed color set have a time stamp and they can only be used if the value of the time stamp is less than the value of the global clock.

A step that can be taken in the eEPC corresponds to a transition firing, given preconditions and postconditions expressed as expressions on arcs or guards (boolean expressions) on transitions. The global clock (model time) advances the time stamps of all timed tokens with the smallest amount of time needed to make at least one token active, which corresponds to the time step rule in eEPCs in which timers decrease their values. Token delays are positioned on transitions or on outgoing arcs of transitions.

5.4.1 Transformation of eEPCs into TCPNs

To map eEPCs into TCPNs, we first identify generic eEPC patterns and provide their translation into TCPN patterns, and then we show how the obtained TCPN patterns can be fused together to form a TCPN.

We define eEPC patterns taking into account the semantic rules given in Section 5.3, covering all patterns that would allow us to build an arbitrary eEPC. An eEPC pattern consists of incoming and outgoing arc(s) and other elements necessary for the corresponding rule.

First we give some notations and notions necessary for the mappings of the elements of an eEPC to elements of timed colored Petri nets. Specific elements will be defined within the patterns.

We denote the non-timed integer color corresponding to the set of process folders numbers with time stamp $\perp$ by PF, the timed integer color corresponding to the set of process folders numbers with time stamp values in $\mathbb{N}$ by PFT, and for each capacity resource and data attribute, we define a place having the corresponding non-timed color type. The locations at which process folders can reside correspond to TCPN places having type PF and local steps in TCPNs are depicted by transitions.

In what follows we describe the TCPN patterns obtained from eEPC patterns in more detail and define a mapping h from eEPC elements to TCPN elements. The left hand sides of Figures 22, 23 and 24 show instances of these patterns having the incoming (outgoing) arcs dotted and their corresponding instantiation of the TCPN pattern. For each TCPN pattern, we distinguish

input and output places corresponding to the incoming and outgoing arcs of the eEPC pattern.

Definition 5.5. [TRANSLATION OF EEPC PATTERNS TO TCPN PATTERNS] *Let G_e be an arbitrary eEPC and $U = \langle \mathbb{S}, \Sigma, \rightarrow, s_0 \rangle$ the transition system describing the semantics of G_e. Let $r \in \mathcal{R}$, $\bar{r} \in \bar{R}$ and $a \in \mathcal{A}$. We define $h(r)$ as p_r, $cd(p_r) = Type(r)$, $h(\bar{r})$ as $p_{\bar{r}}$, $cd(p_{\bar{r}}) = Type(\bar{r})$, $h(a)$ as p_a and $cd(p_a) = Type(a)$ and call the places p_r, $p_{\bar{r}}$ and p_a* global *places.*

start event set *Let E_s^d be a start event set for some $d \in I_s$. The corresponding TCPN pattern is a TCPN represented by an input place p_d with $cd(p_d) =$ PFT which is initially marked with a token $d@0$; a transition t_d with $l(t_d) = E_s^d$ and a set $P_d^{out} = \{p_e^{out} | e \in E_s^d\}$ of output places corresponding to the outgoing arcs of the start events of the start event set so that $cd(p_e^{out}) =$ PF. We define*

- *$h(E_s^d)$ as p_d, $h(a_e^{out})$ as p_e^{out}, for all $p_e^{out} \in P_d^{out}$;*
- *${}^\bullet t_d = [p_d]$, $t_d^\bullet = [p_d] + \sum_{e \in E_s^d} [p_e^{out}]$;*
- *$\mathcal{E}(p_d, t_d) = i$, $\mathcal{E}(t_d, p_e^{out}) = i$, for all $p_e^{out} \in P_d^{out}$ and $\mathcal{E}(t_d, p_d) = (i + |I_s|)@+\iota_d$, where $\iota_d \in Dom(pdf_d)$ and i is a variable of type* PF.

Figure 22(a) shows an instantiation of the eEPC pattern for a start event set with two start events and the corresponding instantiation of the TCPN pattern.

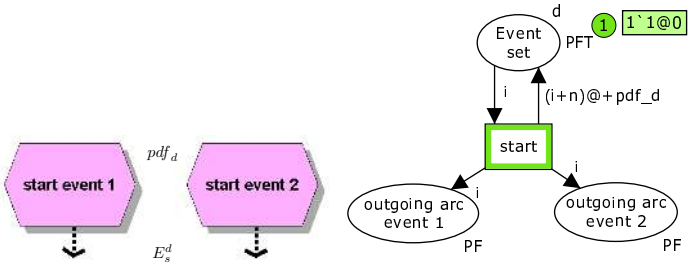
event *Let $e \in E \setminus (E_s \cup E_f \cup E_c)$ be an event. The corresponding TCPN pattern consists of an input place p_e^{in} with $cd(p_e^{in}) =$ PF, a transition t_e with $l(t_d) = e$ and an output place p_e^{out} with $cd(p_e^{out}) =$ PF. An instantiation of it is shown in Figure 22(b). We define*

- *$h(e)$ as t_e, $h(a_e^{in})$ as p_e^{in}, $h(a_e^{out})$ as p_e^{out};*
- *${}^\bullet t_e = [p_e^{in}]$ and $t_e^\bullet = [p_e^{in}]$;*
- *$\mathcal{E}(p_e^{in}, t_e) = \mathcal{E}(t_e, p_e^{out}) = i$, where i is a variable of type* PF.

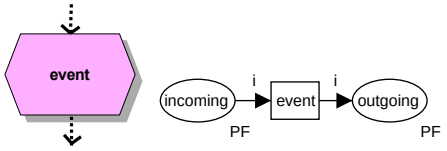
Note that final events are not represented in the translation.

function *Let f be a (timed) function connected to the set of resources $\mathcal{R}_f$ and to the set of data attributes $\mathcal{A}_f$. The corresponding TCPN pattern has an input place (p_f^{in}), two transitions — t_f^a with $l(t_f^a) = f_a$ and t_f^r with $l(t_f^r) = f_r$ — corresponding to the two steps of the rule, an intermediate place (p_f^{int}) and one output place (p_f^{out}). The operations and conditions on the resources and attributes are described on the arc inscriptions and transition guards, respectively. We define $h(a_f^{in})$ as p_f^{in}, $h(a_f^{out})$ as p_f^{out} and $h(f)$ as p_f^{int}. We set*

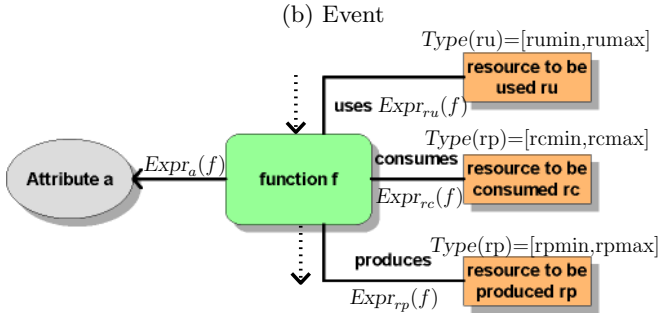
- *$cd(p_f^{in}) = p_f^{out} =$ PF;*
- *$cd(p_f^{int}) =$ PF $\times \Pi_{x \in \mathcal{R}_f \cup \mathcal{A}_f} Type(x)$ timed;*
- *$Gd(t_f^a) = \bigwedge_{r \in \mathcal{R}_f^u \cup \mathcal{R}_f^c} (r - Expr_r(f) \geq r_{\min}) \wedge \bigwedge_{\bar{r} \in \bar{\mathcal{R}}_f^p} (\bar{r} + Expr_r(f) \leq r_{\max})$;*
- *${}^\bullet t_f^a = [p_f^{in}] + \sum_{x \in \mathcal{A}_f \cup \mathcal{R}_f^u \cup \mathcal{R}_f^c \cup \bar{\mathcal{R}}_f^c \cup \bar{\mathcal{R}}_f^p} [h(x)]$;*
- *$t_f^{a\bullet} = [p_f^{int}] + \sum_{x \in \mathcal{A}_f \cup \mathcal{R}_f^u \cup \mathcal{R}_f^c \cup \bar{\mathcal{R}}_f^c \cup \bar{\mathcal{R}}_f^p} [h(x)]$;*
- *${}^\bullet(t_f^r) = [p_f^{int}] + \sum_{x \in \mathcal{A}_f \cup \mathcal{R}_f^p \cup \mathcal{R}_f^u} [h(x)]$;*

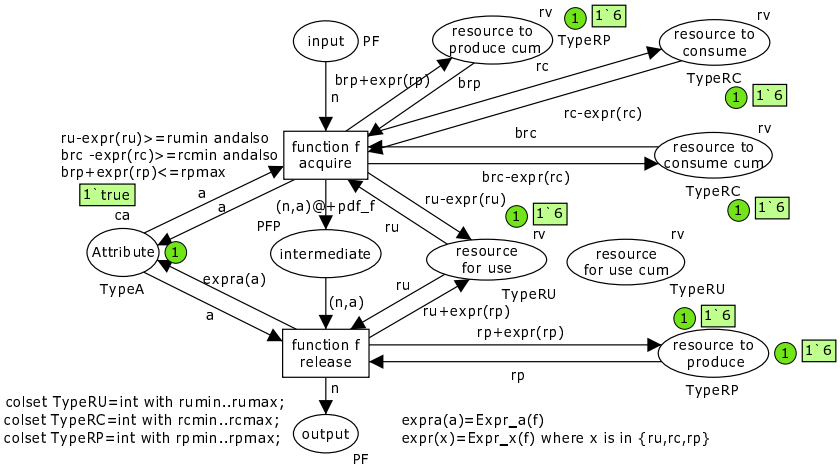


(a) Start event set



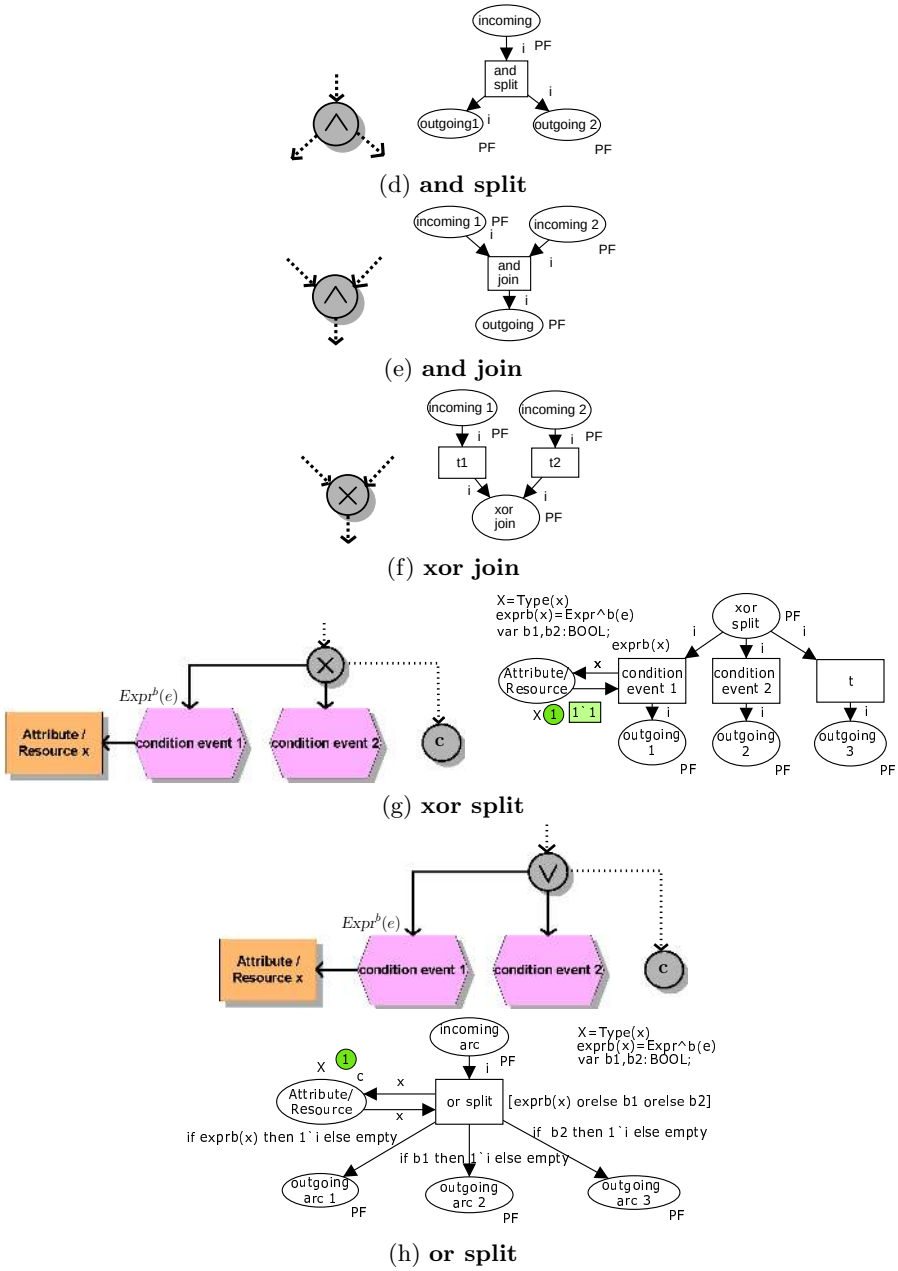
(b) Event





(c) Function

Fig. 22: Translation of the eEPC patterns into TCPN patterns (a-c)



(d) **and split**

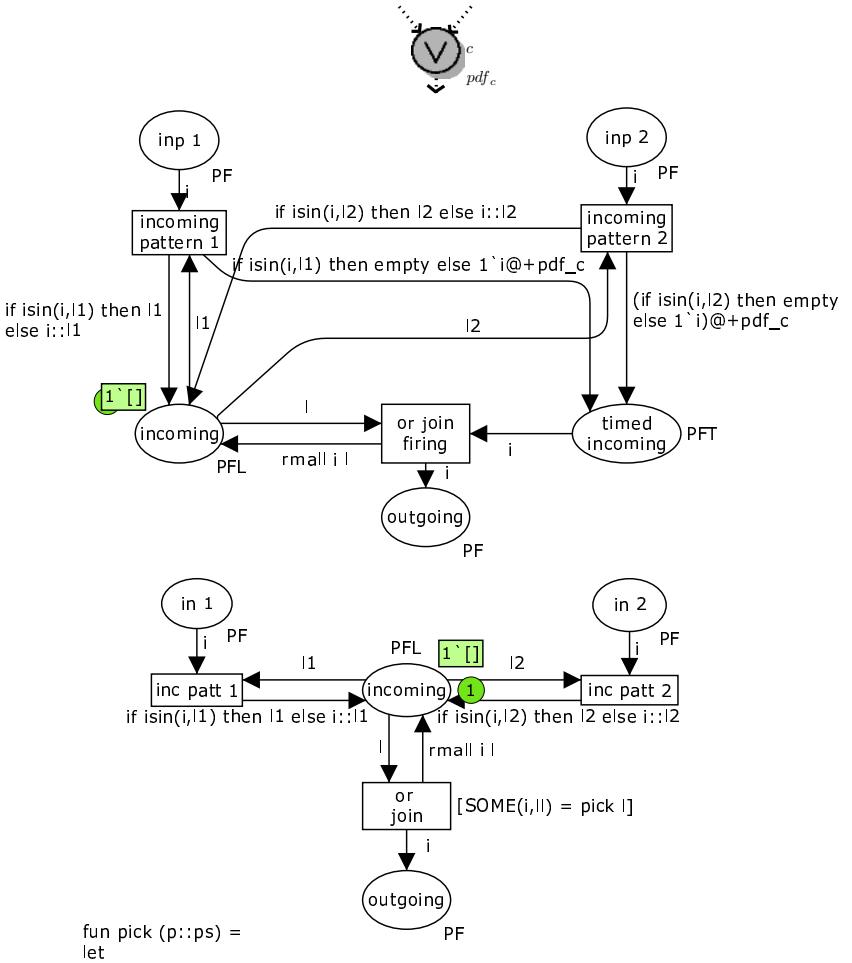
(e) **and join**

(f) **xor join**

(g) **xor split**

(h) **or split**

Fig. 23: Translation of the eEPC patterns into TCPN patterns (d-h)




```
fun pick (p::ps) =
let
val i = discrete (0, (List.length (p::ps)) - 1)
in
SOME (List.nth ((p::ps), i), List.take ((p::ps), i) ^^ List.drop ((p::ps), i+1))
end
| pick [] = NONE;

fun isin(m,[])=false | isin(m,n::l)= if m=n
then true else isin(m,l);
```

Fig. 24: **or join** (timed and non-timed)

- $(t_f^r)^\bullet = [p_f^{out}] + \sum_{x \in \mathcal{A}_f \cup \mathcal{R}_f^p \cup \mathcal{R}_f^u} [h(x)]$;
- $\mathcal{E}(t_f^a, h(r)) = r - Expr_r(f)$ *and* $\mathcal{E}(h(r), t_f^a) = r$ *for all* $r \in \mathcal{R}_f^c \cup \mathcal{R}_f^u$;
- $\mathcal{E}(t_f^a, h(\bar{r})) = \bar{r} - Expr_r(f)$ *and* $\mathcal{E}(h(\bar{r}), t_f^a) = \bar{r}$ *for all* $\bar{r} \in \bar{\mathcal{R}_f^c}$;
- $\mathcal{E}(t_f^a, h(r)) = \bar{r} + Expr_r(f)$ *and* $\mathcal{E}(h(\bar{r}), t_f^a) = \bar{r}$ *for all* $\bar{r} \in \bar{\mathcal{R}_f^p}$;
- $\mathcal{E}(t_f^r, h(a)) = Expr_a(f)$, *for all* $a \in \mathcal{A}_f$;
- $\mathcal{E}(t_f^r, h(r)) = r + Expr_r(f)$ *for all resources* $r \in \mathcal{R}_f^p \cup \mathcal{R}_f^u$;
- $\mathcal{E}(h(x), t_f^r) = x$ *for all* $x \in \mathcal{R}_f^p \cup \mathcal{R}_f^u \cup \mathcal{A}_f$;
- $\mathcal{E}(p_f^{in}, t_f^a) = \mathcal{E}(t_f^r, p_f^{out}) = i$, *where* i *is a variable of type* PF;
- $\mathcal{E}(t_f^a, p_f^{int}) = x@ + \iota_f$ *and* $\mathcal{E}(t_f^a, p_f^{int}) = x$, *where* $\iota_f \in Dom(pdf_f)$ *and* $Type(x) = \mathsf{PF} \times \Pi_{a \in \mathcal{A}_f} Type(a)$ *timed.*

Figure 22(c) shows an instantiation of the pattern operating on an attribute, a resource that is used, a resource that is produced and a resource that is consumed and its translation. Each resource is translated as two places (e.g. resource used, resource used com *correspond to the the resource used by the function, each keeping the values of the resource variables)*

and split, and join, xor join *These eEPC patterns are parametrized by the number of the outgoing and incoming arcs, respectively. Their counterpart TCPN patterns are parametrized by the number of input and output places/transitions, respectively.*

$c \in C_{\mathbf{aj}}$ *The TCPN pattern consists of a set of input places set* $P_c^{in} = \{p_a^{in} | a \in A_c^{in}\}$, *a transition* t_c *and an output place* p_c^{out}. *We define* $h(a_c^{out})$ *as* p_c^{out}, $h(c)$ *as* t_c, $l(t_c) = c$, $h(a)$ *as* p_a^{in} *for all* $a \in A_c^{in}$. *We set*

- $cd(p_a^{in}) = \mathsf{PF}$, $(p_a^{in})^\bullet = [t_c]$, $\mathcal{E}(p_a^{in}, t_c) = i$, *for all* $a \in A_c^{in}$,
- $cd(p_a^{out}) = \mathsf{PF}$, $t_c^\bullet = [p_c^{out}]$ *and* $\mathcal{E}(t_c, p_c^{out}) = i$, *where* i *is a variable of type* PF.

$c \in C_{\mathbf{as}}$ *The TCPN pattern consists of a set of output places set* $P_c^{out} = \{p_a^{out} | a \in A_c^{out}\}$, *a transition* t_c *and an input place* p_c^{in}. *We define* $h(a_c^{out})$ *as* p_c^{out}, $h(c)$ *as* t_c, $h(a)$ *as* p_a^{out}, *for all* $a \in A_c^{out}$. *We set*

- $l(t_c) = c$;
- $cd(p_a^{out}) = \mathsf{PF}$, $^\bullet(p_a^{out}) = [t_c]$ *and* $\mathcal{E}(t_c, p_a^{out}) = i$, *for all* $a \in A_c^{out}$;
- $cd(p_c^{in}) = \mathsf{PF}$, $^\bullet t_c = [p_c^{in}]$ *and* $\mathcal{E}(p_c^{in}, t_c) = i$, *where* i *is a variable of type* PF.

$c \in C_{\mathbf{xj}}$ *The TCPN pattern consists of a set of input places* $P_c^{in} = \{p_a^{in} | a \in A_c^{in}\}$, *a set of transitions* $T_c = \{t_a^{in} | a \in A_c^{in}\}$ *and an output place* p_c^{out}. *We define*

- $h(a) = p_a^{in}$ *for all* $a \in A_c^{in}$, $h(a_c^{out}) = p_c^{out}$, $h(c)$ *as* T_c;
- $l(t_a^{in}) = c$;
- $cd(p_a^{in}) = \mathsf{PF}$, $^\bullet(t_a^{in}) = [p_a^{in}]$, $\mathcal{E}(p_a^{in}, t_a^{in}) = i$, *where* i *is a variable of type* PF;
- $cd(p_c^{out}) = \mathsf{PF}$, $(t_a^{in})^\bullet = [p_c^{out}]$ *and* $\mathcal{E}(t_a^{in}, p_c^{out}) = i$ *for all* $a \in A_c^{in}$, *where* i *is a variable of type* PF.

Figure 23(d-f) shows the respective instantiations of the patterns with two incoming, respectively outgoing arcs and their corresponding instantiations of the TCPN patterns;

xor split *Let c be an* **xor split** *connector and let $A_c = \{(c,x)|x \in (E_f \cap E_c) \cup F \cup C\} \cup \{(e,x)|e \in c^\bullet \cap (E_c \setminus E_f)\}$ and $A' = \{a'|(a' = (c,e) \in A_c \vee a' = (e,x) \in A_c) \wedge e \in E \wedge Var(Expr^b(e)) \neq \varnothing\}$. The TCPN pattern consists of an input place p_c^{in}, a set of output places $P_c^{out} = \{p_a^{out}|a \in A_c\}$, and a set of transitions $T_c^{out} = \{t_a^{out}|a \in A_c\}$ so that $l(t_a^{out}) = c$, for all $t_a^{out} \in T_c^{out}$. The boolean expressions on the condition events of c (on resources or attributes) become guards for the outgoing transitions. We define for any* **xor split** *connector c:*

- $h(a_c^{in}) \stackrel{\text{def}}{=} p_c^{in}$;
- $h(a)$ *as* p_a^{out}, *for all* $a \in A_c$;
- $h(c)$ *as* T_c^{out};
- $cd(p_a^{in}) = \mathsf{PF}$, $(p_a^{in})^\bullet = \sum_{t_a^{out} \in T_c^{out}} [t_a^{out}]$, $\mathcal{E}(p_a^{in}, t_a^{out}) = i$ *for all* $t_a^{out} \in T_c^{out}$;
- $cd(p_a^{out}) = \mathsf{PF}$, ${}^\bullet(p_a^{out}) = [t_a^{out}]$, $\mathcal{E}(t_a^{out}, p_a^{out}) = i$, *for all* $p_a^{out} \in P_c^{out}$;
  - *For all* $a \in A'$, *we set*
    - $Gd(t_a^{out}) = Expr^b(e)$;
    - $(t_a^{out})^\bullet = [p_a^{out}] + \sum_{x \in Var(Expr^b(e))} [h(x)]$;
    - ${}^\bullet(t_a^{out}) = [p_c^{in}] + \sum_{x \in Var(Expr^b(e))} [h(x)]$;
    - $\mathcal{E}(h(x), t_a^{out}) = \mathcal{E}(t_a^{out}, h(x)) = x$ *for all* $x \in Var(Expr^b(e))$.
  - *For all* $a \in A_c \setminus A'$, *we set* $(t_a^{out})^\bullet = [p_a^{out}]$ *and* ${}^\bullet(t_a^{out}) = [p_c^{in}]$.

Figure 23(g) shows an **xor split** *connector having a condition event linked to an attribute or resource, a condition event with condition true and a connector on the outgoing arcs and the corresponding TCPN translation.*

or split *Let c be an* **or split** *connector and let $A_c = \{(c,x)|x \in (E_f \cap E_c) \cup F \cup C\} \cup \{(e,x)|e \in c^\bullet \cap (E_c \setminus E_f)\}$ and $A' = \{a'|(a' = (c,e) \in A_c \vee a' = (e,x) \in A_c) \wedge e \in E \wedge Var(Expr^b(e)) \neq \varnothing\}$. The TCPN pattern contains an in place p_c^{in}, a transition t_c, $l(t_c) = c$ and a set of outgoing places $P_c^{out} = \{p_a^{out}|a \in A_c\}$.*

We define $h(c)$ as t_c, $h(a_c^{in}) = p_c^{in}$ and $h(a)$ as p_a^{out}, for all $a \in A_c$ for any **or split** *connector c.*

- *We set* $cd(p_c^{in}) = \mathsf{PF}$, $cd(p_c^{out}) = \mathsf{PF}$ *for all* $p_a^{out} \in P_c^{out}$.
- *For all* $a \in A'$, *we set* $\mathcal{E}(t_c, h(a)) = \mathsf{if}\ Expr^b(e)\ \mathsf{then}\ 1`i\ \mathsf{else}\ \mathsf{empty}$.
- *For all* $a \in A_c \setminus A'$, *we set* $\mathcal{E}(t_c, h(a)) = \mathsf{if}\ b_a\ \mathsf{then}\ 1`i\ \mathsf{else}\ \mathsf{empty}$, *where b_a is a boolean variable.*
- *Let* $X = \{x \in \mathcal{R} \cup \mathcal{A} | \exists e \in E_c \cap c^\bullet \wedge x \in Var(Expr^b(e))\}$.
  - *For each* $x \in X$, *we set* $\mathcal{E}(h(x), t_c) = \mathcal{E}(t_c, h(x)) = x$.
  - *We set* $(t_c)^\bullet = \sum_{p \in P_c^{out}} [p] + \sum_{x \in X} [h(x)]$.
  - *We set* ${}^\bullet(t_c) = [p_c^{in}] + \sum_{x \in X} [h(x)]$.
- *We set* $\mathcal{E}(p_c^{in}, t_c) = i$ *and* $\mathcal{E}(t_c, p_a^{out}) = i$ *for all* $p_a^{out} \in P_c^{out}$, *where i is a variable of type* PF.

Figure 23(h) shows an instance of the pattern and its translation. In case at least one of the boolean expressions corresponding to conditions on attributes or resources is true, a boolean variable generated for each arc that does not lead to a condition event with conditions on attributes or resources.

In case all the boolean expressions on attributes or resources are false or there are no condition events on data attributes or resources, the boolean variables generated for the rest of the arcs must have at least one true value. The boolean expressions on the condition events and the boolean values generated become conditions in the arc inscriptions.

or join *The TCPN pattern for an* **or join** *connector c contains a transition t_c so that $l(t_c) = c$, an output place p_c^{out}, an input place p_c^u which collects in a list one identifier for each token arriving by firing the output transitions of the adjacent pattern and an additional input place p_c^t which collects timed tokens for the case the* **or join** *connector is non-timed.*

- *We define $h(c)$ as t_c and $h(a_c^{out})$ as p_c^{out}, and*
  - *if $c \in C_t$, we set*
    - $h(A_c^{in}) \stackrel{\text{def}}{=} \{p_c^u, p_c^t\}$, $Type(p_c^u) = \mathsf{PFL}$[6] *and* $Type(p_c^t) = \mathsf{PFT}$;
    - ${}^\bullet t_c = [p_c^u] + [p_c^t]$ *and* $t_c^\bullet = [p_c^u] + [p_c^{out}]$, $\mathcal{E}(p_c^u, t_c) = l$, $\mathcal{E}(t_c, p_c^u) = \mathsf{rmall}\ i\ l$ *and* $\mathcal{E}(p_c^t, t_c) = i$, *where i is a variable of type* PFT *and l is a variable of type* PFL;
    - $Gd(t_c) = \mathsf{SOME}(i, ll) = \mathsf{pick}\ \mathsf{l}$, *for some variable ll of type* PFL.
  - *if $c \in C_{\mathbf{oj}} \setminus C_t$, we set*
    - $h(A_c^{in}) \stackrel{\text{def}}{=} p_c^u$ *and* $Type(p_c^u) = \mathsf{PFL}$;
    - ${}^\bullet t_c = [p_c^u]$, $\mathcal{E}(p_c^u, t_c) = l$, $\mathcal{E}(t_c, p_c^u) = \mathsf{rmall}\ i\ l$ *and* $t_c^\bullet = [p_c^u] + [p_c^{out}]$, *where i is a variable of type* PFT *and l is a variable of type* PFL.
- *We set $\mathcal{E}(t_c, p_c^{out}) = i$.*

Figure 24 shows an instantiation of the eEPC pattern with two incoming arcs and its translation.

Two instantiations of eEPC patterns are called *adjacent* if an outgoing arc of a pattern coincides with an incoming arc of the other pattern. Given an eEPC G_e, we can decompose it into instantiations of eEPC patterns such that every instantiation used in the decomposition is adjacent to at least one other instantiation of an eEPC pattern. Two instantiations of the TCPN patterns are called *adjacent* if they are the translations of two adjacent instantiations of eEPC patterns. An instantiation of a eEPC pattern is an *input* of an instantiation of the **or join** pattern if there exists an outgoing arc of the former which is an incoming arc for the instantiation of an **or join** pattern.

Building a TCPN translation out of the instantiations of the TCPN patterns consists of two steps. First, adjacent instantiations of non-**or join** TCPN patterns are connected, i.e. for each common arc of the corresponding instantiations of the eEPC patterns, if the input place and the output place corresponding to the arc are the same, then the two places are fused. Subsequently, ingoing instantiations of an instantiation of every **or join** pattern are modified accordingly. For each such an ingoing instantiation, the place corresponding to

[6] PFL = list PF is a process folder list type, [] is the empty list and $i :: l$ denotes the concatenation of the element i to the list l

the outgoing arc of the ingoing instantiation adjacent to the instantiation of the **or join** pattern is deleted and arc(s) with corresponding arc expression are added between the input transition of that places and the incoming place(s) of the instantiation of the **or join** pattern.

The translation of G_e is a TCPN $\mathcal{N}_e = \langle P, T, F, \mathcal{E}, Gd, \mathcal{C}, cd, \mathcal{V}, l\rangle$, where $\mathcal{C} = \{Type(x) | x \in \mathcal{R} \cup \mathcal{A}\} \cup \mathsf{PF} \cup \mathsf{PFT} \cup \mathsf{PFL} \cup \mathsf{PFP}$ is the set of colors (types) that includes the resource types and data attribute types, untimed process folder color (PF), timed process folder color (PFT), process folder list color (PFL), $\mathsf{PFP} = \bigcup_{f \in F: \mathcal{R}_f \cup \mathcal{A}_f \neq \varnothing} \mathsf{PF} \times \Pi_{x \in \mathcal{R}_f \cup \mathcal{A}_f} Type(x)$ and $\mathcal{V}$ denotes the finite set of variables.

Let $x \in \mathsf{PFP}$. We denote the projection on the process folder value by $\pi_1(x) \in \mathbb{N}$ and the projection on the value of an attribute $a \in \mathcal{A}$ by $\pi_a(x) \in Type(a)$.

Let G_e be an eEPC and $\mathcal{N}_e$ its TCPN translation. Let $U = \langle \mathbb{S}, \Sigma, \rightarrow, s_0\rangle$ be the transition system of the eEPC G_e and $U' = \langle \mathfrak{S}(\mathcal{N}_e, \mathfrak{s}_0), \Sigma \setminus \{c_w | c \in C_t\}, \longrightarrow, \mathfrak{s}_0\rangle$ the transition system of the TCPN $\mathcal{N}_e$. The initial state of $\mathcal{N}_e$ is $\mathfrak{s}_0 = (M_0, 0)$, where M_0 is

$$\sum_{d \in I_s} [((h(E_s^d), d), 0)] + \sum_{x \in \mathcal{R} \cup \overline{\mathcal{R}} \cup \mathcal{A}} [(p_x, Val_0(x))] + \sum_{c \in C_{\mathbf{oj}} \wedge p_c^u \in h(A_c^{in})} [(p_c^u, [])].$$

We define a transition relation $\Rightarrow \subseteq \mathbb{S} \times \mathbb{S}$ considering **or join** waiting steps in the eEPC transition relation from Definition 5.4 as invisible actions (τ steps). We write $s \overset{x}{\Rightarrow} s'$ iff $s \overset{\tau^*}{\rightarrow} s_1' \overset{x}{\rightarrow} s_2' \overset{\tau^*}{\rightarrow} s'$ for $x \in \Sigma \setminus \{c_w | c \in C_t\}$.

We consider no τ steps in the TCPN, i.e. $\Longrightarrow = \longrightarrow$.

Let $U = \langle \mathbb{S}, \Sigma', \Rightarrow, s_0\rangle$ be the transition system of the eEPC G_e and $U' = \langle \mathfrak{S}(\mathcal{N}_e, \mathfrak{s}_0), \Sigma', \Longrightarrow, \mathfrak{s}_0\rangle$ the transition system of the TCPN $\mathcal{N}_e$, where $\Sigma' = \Sigma \setminus \{c_w | c \in C_t\}$.

We call a process folder $\mathfrak{f} = (v, (i, d))$ *non-waiting* iff $v \in \{a_v^{in} | v \in F \cup C_{\mathbf{s}} \cup (E_f \cup (E_i \setminus E_c))\} \cup \{a \in A_v^{in} | v \in C_{\mathbf{xj}} \cup C_{\mathbf{aj}}\} \cup F$.

Similarly, we define *non-waiting case tokens* as tokens $\nu = ((p, x), \iota)$, where $i \in \mathbb{N}$, $\iota \in \mathbb{N}^+$, $p = h(v)$ and either

- $v \in \{a_v^{in} | v \in F \cup C_{\mathbf{s}} \cup (E_f \cup (E_i \setminus E_c))\} \cup \{a \in A_v^{in} | v \in C_{\mathbf{xj}} \cup C_{\mathbf{aj}}\}$ and $x \in \mathbb{N}$, or
- $v \in F$, $\pi_1(x) \in \mathbb{N}$ and $\pi_a(x) \in Type(a)$ for all $a \in \mathcal{A}_f$.

Now we shall show that the transformation defined preserves the behavior up to weak bisimulation. We define a relation $\mathcal{S}$ between a state of the eEPC and a state of the TCPN.

Let $s = (m, Val)$ be an eEPC state and $\mathfrak{s} = (M, \iota)$ a TCPN state. We first define the relation $\mathcal{S}$ between non-waiting process folders $(x, (i, n)) \in m$ and non-waiting case tokens $((p, i), \iota) \in M$, for some $i \in \mathbb{N}$, $n \in \mathbb{N}_\perp$, global time $\iota' \in \mathbb{N}$ and time stamp $\iota \in \mathbb{N}$. We write $(x, (i, n)) \; \mathcal{S} \; ((p, i), \iota)$ if

- the location of the process folder corresponds to the location of the token, i.e. $h(x) = p$;
- the time stamp of the token and the timer of the process folder agree:

- $x \in \{a_v^{in} | v \in F \cup C_{\mathbf{s}} \cup (E_f \cup (E_i \setminus E_c))\} \cup \{a \in A_v^{in} | v \in C_{\mathbf{xj}} \cup C_{\mathbf{aj}}\}$, $n = \bot$ and $\iota' = \iota$ or
- $x = f$ for some $f \in F$, $n = 0$ and $\iota' = \iota$, or
- $x = f$ for some $f \in F$, $n > 0$ and $n + \iota' = \iota$.

Lemma 5.2. *Let G_e be an eEPC, $\mathfrak{F}$ the set of all non-waiting process folders of G_e, $\mathcal{N}_e$ its TCPN translation and $\mathfrak{T}$ the set of all non-waiting case tokens of G_e. The relation $\mathcal{S}$ between non-waiting process folders of G_e and non-waiting case tokens $\mathcal{N}_e$ is a one-to-one relation, i.e. for all $\mathfrak{f} \in \mathfrak{F}$ there exists exactly one $\nu \in \mathfrak{T}$ such that $\mathfrak{f}\ \mathcal{S}\ \nu$ and for all $\nu \in \mathfrak{T}$ there exists exactly one $\mathfrak{f} \in \mathfrak{F}$ such that $\mathfrak{f}\ \mathcal{S}\ \nu$.*

Lemma 5.3. *Let G_e be an eEPC and $\mathcal{N}_e$ its TCPN translation and $U' = \langle \mathfrak{S}(\mathcal{N}_e, \mathfrak{s}_0), \Sigma', \Longrightarrow, \mathfrak{s}_0 \rangle$ the transition system of $\mathcal{N}_e$. Then for all reachable states $(M, \iota) \in \mathfrak{S}(\mathcal{N}_e, \mathfrak{s}_0)$, for all non-timed* **or join** *connectors $c \in C_{\mathbf{oj}} \setminus C_t$, $M(h(A_c^{in}), l) = 1$ and for all timed* **or join** *connectors $c \in C_t$, $M(p_c^u, l) = 1$, $\{i | i \in l\} = \{i | ((p_c^t, i), \iota') \in M\}$, where $h(A_c^{in}) = \{p_c^u, p_c^t\}$.*

Lemma 5.4. *Let G_e be an eEPC and $\mathcal{N}_e$ its TCPN translation and $U' = \langle \mathfrak{S}(\mathcal{N}_e, \mathfrak{s}_0), \Sigma', \Longrightarrow, \mathfrak{s}_0 \rangle$ the transition system of $\mathcal{N}_e$. Then, at every reachable marking, there is exactly one token on each place corresponding to a start event set, i.e. for all reachable states $(M, \iota) \in \mathfrak{S}(\mathcal{N}_e, \mathfrak{s}_0)$, we have that $\sum_{((h(E_s^d), x), \iota') \in M} M((h(E_s^d), x), \iota') = 1$, for all $d \in I_s$.*

Lemma 5.5. *Let G_e be an eEPC and $\mathcal{N}_e$ its TCPN translation and $U' = \langle \mathfrak{S}(\mathcal{N}_e, \mathfrak{s}_0), \Sigma', \Longrightarrow, \mathfrak{s}_0 \rangle$ the transition system of $\mathcal{N}_e$. Then all global places contain exactly one token at every reachable marking, i.e. for all reachable states $(M, \iota) \in \mathfrak{S}(\mathcal{N}_e, \mathfrak{s}_0)$ and for all $x \in \mathcal{R} \cup \bar{\mathcal{R}} \cup \mathcal{A}$, $\sum_{(h(x), x) \in M} M(h(x), x) = 1$.*

To handle **or join** connectors, we define a family of functions $\bigcup_{c \in C_t} g_c$ that maps a multiset of time stamps to a time stamp, i.e. $g_c : (\mathbb{N}^{\mathbb{N} \cup \{\bot\}} \setminus \varnothing) \to \mathbb{N}$, where:

$$g_c(x) \stackrel{\text{def}}{=} \begin{cases} \iota_c \in Dom(pdf_c) & \bot \in x \text{ and } \forall d \in \mathbb{N} : x(d) = 0; \\ n & \text{if } n \in x \text{ and } \forall m \in x : m \leq n. \end{cases}$$

Now we lift $\mathcal{S}$ to the states of eEPCs and TCPNs accordingly. Let $s = (m, Val)$ be an eEPC state and $\mathfrak{s} = (M, \iota)$ a TCPN state. We write $s\ \mathcal{S}\ \mathfrak{s}$ iff the following conditions are satisfied:

- for all non-waiting case tokens $\nu \in M$ there exists a non-waiting process folder $\mathfrak{f} \in m$ so that $\nu\ \mathcal{S}\ \mathfrak{f}$ and $M(\nu) = m(\mathfrak{f})$, and for all non-waiting process folders $\mathfrak{f} \in m$ there exists a non-waiting case token $\nu \in M$ so that $\nu\ \mathcal{S}\ \mathfrak{f}$;
- for all $x \in \mathcal{R} \cup \bar{\mathcal{R}} \cup \mathcal{A}$, $M(h(x), Val(x)) = 1$;
- for all $d \in I_s$, $\sum_{e \in E_S^d} m(e, (i, \iota_d)) = 1$, $M((h(E_s^d), i), \iota') = 1$ and $\iota' = \iota + \iota_d$, for some $\iota_d \in Dom(pdf_d)$ and $i \in \mathbb{N}$;

– for all timed **or join** connectors $c \in C_t$, and $i \in \mathbb{N}$ such that $(p_c^u, l) \in M$ and $((p_c^t, i), \iota') \in M$, where $p_c^u, p_c^t \in h(A_c^{in})$ and $i \in l$, we have that

$$\mathfrak{F}_{\mathfrak{c},\mathfrak{i}} \stackrel{\text{def}}{=} \sum_{(a,(i,d))\in m \wedge a \in A_c^{in}} [(a,(i,d))] \neq \varnothing,$$

and $g_c(\mathfrak{T}_{c,i}) = \iota' - \iota$, where $\mathfrak{T}_{c,i} = \sum_{((a,i),d)\in \mathfrak{F}_{\mathfrak{c},\mathfrak{i}} \wedge a \in A_c^{in}} [d]$ and for all timed **or join** connectors $c \in C_t$, and $i \in \mathbb{N}$ such that

$$\mathfrak{F}_{\mathfrak{c},\mathfrak{i}} \stackrel{\text{def}}{=} \sum_{(a,(i,d))\in m \wedge a \in A_c^{in}} [(a,(i,d))] \neq \varnothing,$$

we have that $(p_c^u, l) \in M$ and $((p_c^t, i), \iota') \in M$, $p_c^u, p_c^t \in h(A_c^{in})$, $i \in l$ and $g_c(\mathfrak{T}_{c,i}) = \iota' - \iota$, where $\mathfrak{T}_{c,i} = \sum_{((a,i),d)\in \mathfrak{F}_{\mathfrak{c},\mathfrak{i}} \wedge a \in A_c^{in}} [d]$.

– for all non-timed **or join** connectors $c \in C_{\mathbf{oj}} \setminus C_t$, and $i \in \mathbb{N}$ such that $(h(A_c^{in}), l) \in M$ and $i \in l$, we have that

$$\mathfrak{F}_{\mathfrak{c},\mathfrak{i}} \stackrel{\text{def}}{=} \sum_{(a,(i,\perp))\in m \wedge a \in A_c^{in}} [(a,(i,\perp))] \neq \varnothing$$

and for all non-timed **or join** connectors $c \in C_{\mathbf{oj}} \setminus C_t$, and $i \in \mathbb{N}$ such that

$$\mathfrak{F}_{\mathfrak{c},\mathfrak{i}} \stackrel{\text{def}}{=} \sum_{(a,(i,\perp))\in m \wedge a \in A_c^{in}} [(a,(i,\perp))] \neq \varnothing,$$

we have $(p_c^u, l) \in M$, where $p_c^u = h(A_c^{in})$ and $i \in l$.

Note that $s_0 \; \mathcal{S} \; \mathfrak{s}_0$.

Theorem 5.1.
Let G_e be an eEPC, $U = \langle \mathbb{S}, \Sigma', \Rightarrow, s_0 \rangle$ the transition system of G_e, $\mathcal{N}_e$ its TCPN translation and $U' = \langle \mathfrak{S}(\mathcal{N}_e, \mathfrak{s}_0), \Sigma', \Longrightarrow, \mathfrak{s}_0 \rangle$ the transition system of $\mathcal{N}_e$. The relation $\mathcal{S}$ is a weak bisimulation.

Proof. ($\Rightarrow$) Let $s = (m, Val), s' = (m', Val') \in \mathbb{S}$ so that $s \stackrel{x}{\Rightarrow} s'$, for some $x \in \Sigma$ and $\mathfrak{s} = (M, \iota) \in \mathfrak{S}(\mathcal{N}_e, \mathfrak{s}_0)$ so that $s \; \mathcal{S} \; \mathfrak{s}$. Depending on the rule applied, we have the following cases:

event set Since $s \stackrel{d}{\Rightarrow} s'$, there exists a start event set E_s^d and an $i \in \mathbb{N}$ such that $\sum_{e_d \in E_s^d} [(e_d, (i,0))] \in m$ and $m' = m - \sum_{e \in E_s^d} [(e, (i,0))] + \sum_{e \in E_s^d} [(e, (i + |I_s|, \iota_d))] + \sum_{e \in E_s^d} [(a_e^{out}, (i, \perp))]$, for $\iota_d \in Dom(pdf_d)$. Since $s \; \mathcal{S} \; \mathfrak{s}$, there exists $((h(E_s^d), i), \iota) \in M$ and $(e_d, (i,0)) \; \mathcal{S} \; ((h(E_s^d), i), \iota)$.
Then $\mathfrak{s} \stackrel{d}{\Longrightarrow} \mathfrak{s}'$ and

$$M' = M - [((h(E_S^d), i), \iota)] + \sum_{e \in E_d^{out}} M_e + [((h(E_S^d), i + |I_s|), \iota + \iota_d)],$$

where for all $e \in E_d^{out}$, one of the following cases occur:

– $M_e = [((h(a_x^{in}), i), \iota)]$ if $e^\bullet = x \notin C_{\mathbf{oj}}$. Since $s \mathcal{S} \mathfrak{s}$, we have that

$$(m - \sum_{e \in E_s^d} [(e, (i, 0))] + \sum_{e \in E_s^d} [(e, (i + |I_s|, \iota_d))], Val) \mathcal{S}$$
$$(M - [((h(E_S^d), i), \iota)] + [((h(E_S^d), i + |I_s|), \iota + \iota_d)], \iota).$$

Hence $m'(a_e^{out}, (i, \bot)) = M'((h(a_e^{out}), i), \iota) = 1$ and $(a_e^{out}, (i, \bot)) \mathcal{S} ((h(a_e^{out}), i), \iota)$.

– If $e^\bullet = c \in C_{\mathbf{oj}}$ and $i \in l$, where $(p_c^u, l)^\iota \in M$ and $p_c^u \in h(A_c^{in})$, $M_e = \varnothing$.
– If $e^\bullet = c \in C_{\mathbf{oj}}$ and $i \notin l$, where $(p_c^u, l)^\iota \in M$ and $p_c^u \in h(A_c^{in})$,
  - If $c \in C_{\mathbf{oj}} \setminus C_t$, $M_e = [((h(A_c^{in}), i :: l), \iota)] - [((h(A_c^{in}), l), \iota)]$.
  - If $c \in C_t$, $M_e = [((p_c^u, i :: l), \iota)] - [((p_c^u, l), \iota)] + [((p_c^t, i), \iota + \iota_c)]$, where $\iota_c \in Dom(pdf_c)$ and $(a_e^{out}, (i, \bot)) = (a_c^{in}, (i, \bot)) \in m'$.

Since $Val'(x) = Val(x) = M(h(x)) = M'(h(x))$ for all $x \in \mathcal{A} \cup \mathcal{R} \cup \bar{\mathcal{R}}$ and

$$(m - \sum_{e \in E_s^d} [(e, (i, 0))] + \sum_{e \in E_s^d} [(e, (i + |I_s|, \iota_d))], Val) \mathcal{S}$$
$$(M - [((h(E_S^d), i), \iota)] + [((h(E_S^d), i + |I_s|), \iota + \iota_d)], \iota).$$

we have that $s' \mathcal{S} \mathfrak{s}'$.

event Let e be the event such that the event rule $s \overset{e}{\Rightarrow} s'$ is applicable. Since $s \overset{e}{\Rightarrow} s'$, there exists a process folder $(a_e^{in}, (i, \bot)) \in m$ for some $i \in \mathbb{N}$ and $m' = m - [(a_e^{in}, (i, \bot))] + [(a_e^{out}, (i, \bot))]$. Since $s \mathcal{S} \mathfrak{s}$, there exists $((p, i), \iota) \in M$ and $(a_e^{in}, (i, \bot)) \mathcal{S} ((h(a_e^{in}), i), \iota)$.

Then $\mathfrak{s} \overset{e}{\Longrightarrow} \mathfrak{s}'$, where

– if $e^\bullet \notin C_{\mathbf{oj}}$, $M' = M - [(h(a_e^{in}), i), \iota)] + [((h(a_e^{out}), i), \iota)]$. Since $s \mathcal{S} \mathfrak{s}$ and $m(a_e^{out}, (i, \bot)) = M((h(a_e^{out}), i), \iota)$, we have that $m'(a_e^{out}, (i, \bot)) = M'((h(a_e^{out}), i), \iota)$ and $(a_e^{out}, (i, \bot)) \mathcal{S} ((h(a_e^{out}), i), \iota)$.
– If $e^\bullet \in C_{\mathbf{oj}}$ and $i \in l$, where $(p_c^u, l)^\iota \in M$ and $p_c^u \in h(A_c^{in})$, $M_e = \varnothing$.
– If $e^\bullet \in C_{\mathbf{oj}}$ and $i \notin l$, where $(p_c^u, l)^\iota \in M$ and $p_c^u \in h(A_c^{in})$,
  - If $c \in C_{\mathbf{oj}} \setminus C_t$, $M_e = [((h(A_c^{in}), i :: l), \iota)] - [((h(A_c^{in}), l), \iota)]$.
  - If $c \in C_t$, $M_e = [((p_c^u, i :: l), \iota)] - [((p_c^u, l), \iota)] + [((p_c^t, i), \iota + \iota_c)]$, where $\iota_c \in Dom(pdf_c)$ and $(a_c^{in}, (i, \bot)) = (a_e^{out}, (i, \bot)) \in m'$.

  Hence $s' \mathcal{S} \mathfrak{s}'$.

function acquiring Let f be the function such that the a consuming function rule $s \overset{f_a}{\Rightarrow} s'$ is applicable. Since $s \overset{f_a}{\Rightarrow} s'$, there exists a process folder $(a_f^{in}, (i, \bot)) \in m$ for some $i \in \mathbb{N}$, $m' = m - [(a_f^{in}, (i, \bot))] + [(f, (i, \iota_f))]$, where $\iota_f \in Dom(pdf_f)$ and

– $Val'(r) = Val(r) - Expr_r(f)$ for all $r \in \mathcal{R}_f^c \cup \mathcal{R}_f^u$,
– $Val'(\bar{r}) = Val(\bar{r}) - Expr_r(f)$ for all $\bar{r} \in \bar{\mathcal{R}}_f^c$,
– $Val'(\bar{r}) = Val(\bar{r}) + Expr_r(f)$ for all $\bar{r} \in \bar{\mathcal{R}}_f^p$ and $Val'(x) = Val(x)$ for all $x \in (\mathcal{R} \cup \bar{\mathcal{R}} \cup \mathcal{A}) \setminus (\mathcal{R}_f^c \cup \mathcal{R}_f^u \cup \bar{\mathcal{R}}_f^c \cup \bar{\mathcal{R}}_f^p)$.

Since $s \mathcal{S} \mathfrak{s}$, there exists $((h(a_f^{in}), i), \iota) \in M$, a binding b and

– for all $r \in \mathcal{R}_f^u \cup \bar{\mathcal{R}}_f^c$, $(h(r), b(\mathcal{E}(h(r), t_f^a))) \in M$ and $b(\mathcal{E}(h(r), t_f^a) - c_r^f) \geq r_{\min}$;

– for all $r \in \bar{\mathcal{R}}_f^p$, $(h(r), b(\mathcal{E}(h(r), t_f^a)) \in M$ and $\mathcal{E}(h(r), t_f^a) + c_r^f \geq r_{\max}$;
– $(h(f), ia) \in M$ and $\pi_1(b(ia)) = i$ and $Val(a) = \pi_a(b(ia))$ for all $a \in A_f$.

Then, $\mathfrak{s} \xrightarrow{(f_a, b)} \mathfrak{s}'$, i.e. $\mathfrak{s} \xRightarrow{f_a} \mathfrak{s}'$, where

$$\begin{aligned} M' = M &- [((h(a_f^{in}), i), \iota)] + [((h(f), b(ia)), \iota + \iota_f)] + \\ &\sum_{r \in \mathcal{R}_f^c \cup \mathcal{R}_f^u \cup \bar{\mathcal{R}}_f^p} ([(h(r), b(\mathcal{E}(h(r), t_f^a)) - c_r^f)] - [(h(r), b(\mathcal{E}(h(r), t_f^a)))]) + \\ &\sum_{r \in \mathcal{R}_f^c \cup \bar{\mathcal{R}}_f^p} ([(h(r), b(\mathcal{E}(h(r), t_f^a)) - c_r^f)] - [(h(r), b(\mathcal{E}(h(r), t_f^a)))]), \end{aligned}$$

where $\pi_1(b(ia)) = i$ and $\pi_a(b(ia)) = Val(a)$, for all $a \in \mathcal{A}_f$.
Since $Val(r) = b(\mathcal{E}(h(r), t_f^a))$ for all $r \in \mathcal{R}$ and $Val(\bar{r}) = b(\mathcal{E}(h(\bar{r}), t_f^a))$ for all $\bar{r} \in \bar{\mathcal{R}}$, we have

$$\begin{aligned} M' = M &- [((h(a_f^{in}), i), \iota)] + [((h(f), b(ia)), \iota + \iota_f)] + \\ &\sum_{r \in \mathcal{R}_f^c \cup \mathcal{R}_f^u \cup \bar{\mathcal{R}}_f^p} ([(h(r), Val'(r))] - [(h(r), Val(r))]) + \\ &\sum_{r \in \mathcal{R}_f^c \cup \bar{\mathcal{R}}_f^p} ([(h(r), Val'(r))] - [(h(r), Val(r))]). \end{aligned}$$

Hence $Val'(x) = M'(h(x))$, for all $x \in \mathcal{R} \cup \bar{\mathcal{R}} \cup \mathcal{A}$. Since

$$(m - [(a_f^{in}, (i, \bot))], Val) \; \mathcal{S} \; (M - [((h(a_f^{in}), i), \iota)], \iota)$$

and $(f, (i, \iota_f)) \; \mathcal{S} \; ((h(f), b(ia)), \iota + \iota_f)$, we have that $s' \; \mathcal{S} \; \mathfrak{s}'$.

function release Let f be the function such that the a producing function rule $s \xRightarrow{f_r} s'$. Since $s \xRightarrow{f_r} s'$, $(f, (i, 0)) \in m$, for some $i \in \mathbb{N}$, and $m' = m - [(f, (i, 0))] + [(a_f^{out}, (i, \bot))]$, where $Val'(\mathfrak{a}) = eval(Expr_{\mathfrak{a}}(f), Val)$ for all $\mathfrak{a} \in \mathcal{A}_f$, $Val'(r) = Val(r) + Expr_r(f)$ for all resources $r \in \mathcal{R}_f^p \cup \mathcal{R}_f^u$ produced or used, and $Val'(x) = Val(x)$ for all $x \in (\mathcal{R} \cup \bar{\mathcal{R}} \cup \mathcal{A}) \setminus (\mathcal{A}_f \cup \mathcal{R}_f^p \cup \mathcal{R}_f^u)$. Since $s \; \mathcal{S} \; \mathfrak{s}$, there exists a binding b so that $((h(f), b(ia)), \iota) \in M$ and since $(f, (i, 0)) \; \mathcal{S} \; ((h(f), b(ia)), \iota)$, $\pi_1(b(ia)) = i$ and $\pi_a(b(ia)) = Val(a)$, for all $a \in \mathcal{A}_f$.

Then, $\mathfrak{s} \xrightarrow{(f_r, b)} \mathfrak{s}'$, i.e. $\mathfrak{s} \xRightarrow{f_r} \mathfrak{s}'$, where

$$\begin{aligned} M' = M &- [((h(f), b(ia)), \iota)] + \sum_{a \in A_f} [(h(f), b(\mathcal{E}(t_f^r, h(a))))] + M_f + \\ &+ \sum_{r \in \mathcal{R}_f^u \cup \mathcal{R}_f^p} ([(h(r), b(\mathcal{E}(t_f^r, h(r))) + c_r^f)] - [(h(r), b(\mathcal{E}(t_f^r, h(r)))))]), \end{aligned}$$

and
– if $f^\bullet \notin C_{\mathbf{oj}}$, $M_f = [((h(a_f^{out}), i), \iota)]$. Since $(a_f^{out}, (i, \bot)) \; \mathcal{S} \; ((h(a_f^{out}), i), \iota)$ and $m'(a_f^{out}, (i, \bot)) = M'((h(a_f^{out}), i), \iota)$.

- If $f^\bullet \in C_{\mathbf{oj}}$ and $i \in l$, where $(p_c^u, l)^\iota \in M$ and $p_c^u \in h(A_c^{in})$, $M_f = \emptyset$.
- If $f^\bullet \in C_{\mathbf{oj}}$ and $i \notin l$, where $(p_c^u, l)^\iota \in M$ and $p_c^u \in h(A_c^{in})$,
  - If $c \in C_{\mathbf{oj}} \setminus C_t$, $M_f = [((h(A_c^{in}), i :: l), \iota)] - [((h(A_c^{in}), l), \iota)]$.
  - If $c \in C_t$, $M_f = [((p_c^u, i :: l), \iota)] - [((p_c^u, l), \iota)] + [((p_c^t, i), \iota + \iota_c)]$, where $\iota_c \in Dom(pdf_c)$.

Since $Val(r) = b(\mathcal{E}(h(r), t_f^a))$ for all $r \in \mathcal{R}$ and $Val(\bar{r}) = b(\mathcal{E}(h(\bar{r}), t_f^a))$ for all $\bar{r} \in \bar{\mathcal{R}}$, we have

$$M' = M - [((h(f), b(ia)), \iota)] + \sum_{a \in A_f} [(h(f), Val'(a))] + \\ + M_f + \sum_{r \in \mathcal{R}_f^u \cup \mathcal{R}_f^p} ([(h(r), Val'(r))] - [(h(r), Val(r))]),$$

Hence $Val'(x) = M'(h(x))$, for all $x \in \mathcal{R} \cup \bar{\mathcal{R}} \cup \mathcal{A}$. Since

$$(m - [(f, (i, 0))], Val) \; \mathcal{S} \; (M - [((h(f), b(ia)), \iota)], \iota)$$

and $(a_f^{out}, (i, \bot)) \; \mathcal{S} \; ((h(a_f^{out}), i), \iota)$, we have that $s' \; \mathcal{S} \; \mathfrak{s}'$.

and split Let c be an **and split** connector such that the event rule $s \overset{c}{\Rightarrow} s'$ is applicable. Since $s \overset{c}{\Rightarrow} s'$, there exists a process folder $(a_c^{in}, (i, \bot)) \in m$ for some $i \in \mathbb{N}$ and $m' = m - [(a_c^{in}, (i, \bot))] + \sum_{a \in A_c^{out}} [(a, (i, \bot))]$. Since $s \; \mathcal{S} \; \mathfrak{s}$, there exists a token $((h(a_c^{in}), i), \iota) \in M$ and $(a_e^{in}, (i, \bot)) \; \mathcal{S} \; ((a_c^{in}, i), \iota)$. Then $\mathfrak{s} \overset{c}{\Longrightarrow} \mathfrak{s}'$ and $M' = M - [((h(a_c^{in}), i), \iota)] + \sum_{a \in A_c^{out}} M_a$, $\iota' = \iota$, where for all $a \in A_c^{out}$, one of the following cases occur:

- $M_a = [((h(a), i), \iota)]$ if $a = (c, x)$ and $x \notin C_{\mathbf{oj}}$. Hence, we have that $(a, (i, \bot)) \; \mathcal{S} \; ((h(a), i), \iota)$ and $m'(a, (i, \bot)) = M'((h(a), i), \iota)$.
- If $a = (c, x)$ and $x \in C_{\mathbf{oj}}$ and $i \in l$, where $(p_x^u, l) \in M$ and $p_x^u \in h(A_x^{in})$, $M_a = \emptyset$.
- If $a = (c, x)$ and $x \in C_{\mathbf{oj}}$ and $i \notin l$, where $(p_x^u, l) \in M$ and $p_x^u \in h(A_x^{in})$,
  - If $x \in C_{\mathbf{oj}} \setminus C_t$, $M_a = [(h(A_x^{in}), i :: l)] - [(h(A_x^{in}), l)]$.
  - If $x \in C_t$, $M_a = [(p_x^u, i :: l)] - [(p_x^u, l)] + [((p_x^t, i), \iota + \iota_x)]$, where $\iota_x \in Dom(pdf_x)$.

Hence, $s' \; \mathcal{S} \; \mathfrak{s}'$.

xor split Let c be the **xor split** connector for which the **xor split** rule $s \overset{c}{\Rightarrow} s'$ is applicable. Since $s \overset{c}{\Rightarrow} s'$, there exists a process folder $(a_c^{in}, (i, \bot)) \in m$ for some $i \in \mathbb{N}$ and $m' = m - [(a_c^{in}, (i, \bot))] + [(a', (i, \bot))]$, where

- $a' = a_e^{out}$ if $eval(Expr^b(e), Val) = true$ for some event $e \in c^\bullet \setminus E_f$, or
- $a' = (c, e)$, if $eval(Expr^b(e), Val) = true$ for some final event $e \in c^\bullet \cap E_f$, or
- $a' = (c, c')$ for some $c' \in C$.

Since and $s \; \mathcal{S} \; \mathfrak{s}$, there exists $((h(a_c^{in}), i), \iota) \in M$ and for all resources and attributes $x \in \mathcal{R} \cup \mathcal{A}$, $(h(x), Val(x)) \in M$. Hence $b(Expr^b(e)) = eval(Expr^b(e), Val)$, for all condition events $e \in c^\bullet$ for which $Expr^b(e)$ is defined. Then $\mathfrak{s} \overset{c}{\Longrightarrow} \mathfrak{s}'$, where $M' = M - [((h(a_c^{in}), i), \iota)] + M_c$, $\iota' = \iota$, and

- If $a' = (x, y)$ and $y \notin C_{\mathbf{oj}}$, $M_c = [((h(a'), i), \iota)]$. Hence $m'((a', (i, \bot))) = M'((h(a'), i), \iota)$ and $(a', (i, \bot)) \; \mathcal{S} \; ((h(a'), i), \iota)$.
- If $a' = (x, y)$ and $y \in C_{\mathbf{oj}}$ and $i \in l$, where $(p_y^u, l) \in M$ and $p_y^u \in h(A_y^{in})$, $M_c = \varnothing$.
- If $a' = (x, y)$ and $y \in C_{\mathbf{oj}}$ and $i \notin l$, where $(p_y^u, l) \in M$ and $p_y^u \in h(A_y^{in})$,
  - If $y \in C_{\mathbf{oj}} \setminus C_t$, $M_c = [((h(A_y^{in}), i :: l), \iota)] - [((h(A_y^{in}), l), \iota)]$.
  - If $y \in C_t$, $M_c = [((p_y^u, i :: l), \iota)] - [((p_y^u, l), \iota)] + [((p_y^t, i), \iota + \iota_y)]$, where $\iota_y \in Dom(pdf_y)$.

Hence in all cases, we have $s' \; \mathcal{S} \; \mathfrak{s}'$.

or split Let c be the **or split** connector for which the **or split** rule $s \overset{c}{\Rightarrow} s'$ is applicable. Since $s \overset{c}{\Rightarrow} s'$, there exists a process folder $(a_c^{in}, (i, \bot)) \in m$ for some $i \in \mathbb{N}$ and $m' = m - [(a_c^{in}, (i, \bot))] + \sum_{a \in A' \cup A''} [(a, (i, \bot))]$, where $A' \subseteq \{a_e^{out} | e \in c^\bullet \cap (E_c \setminus E_f) \wedge eval(Expr^b(e), Val) = true\}$ and $A'' \subseteq \{(c, e) \in A_c^{out} | e \in c^\bullet \cap (E_c \cap E_f) \wedge eval(Expr^b(e), Val) = true\} \cup \{(c, c') \in A_c^{out} | c' \in C\}$ and $A \cap A'' \neq \varnothing$.

Since and $s \; \mathcal{S} \; \mathfrak{s}$, there exists $((h(a_c^{in}), i), \iota) \in M$. Hence $b(Expr^b(e)) = eval(Expr^b(e), Val)$, for all condition events $e \in c^\bullet$ for which $Expr^b(e)$ is defined. Then, $\mathfrak{s} \overset{c}{\Longrightarrow} \mathfrak{s}'$, where $M' = M - [((h(a_c^{in}), i), \iota)] + \sum_{a' \in A' \cup A''} M_{a'}$, and $\iota' = \iota$, where for all $a' \in A' \cup A''$, one of the following cases occur:

- If $a' = (x, y)$ and $y \notin C_{\mathbf{oj}}$, $M_{a'} = [((h(a'), i), \iota)]$. Hence $m'(a', (i, \bot)) = M'((h(a'), i), \iota)$ and $(a', (i, \bot)) \; \mathcal{S} \; ((h(a'), i), \iota)$.
- If $a' = (x, y)$ and $y \in C_{\mathbf{oj}}$ and $i \in l$, where $(p_y^u, l) \in M$ and $p_y^u \in h(A_y^{in})$, $M_{a'} = \varnothing$.
- If $a' = (x, y)$ and $y \in C_{\mathbf{oj}}$ and $i \notin l$, where $(p_y^u, l) \in M$ and $p_y^u \in h(A_y^{in})$,
  - If $y \in C_{\mathbf{oj}} \setminus C_t$, $M_{a'} = [((h(A_y^{in}), i :: l), \iota)] - [((h(A_y^{in}), l), \iota)]$.
  - If $y \in C_t$, $M_{a'} = [((p_y^u, i :: l), \iota)] - [((p_y^u, l), \iota)] + [((p_y^t, i), \iota + \iota_y)]$, where $\iota_y \in Dom(pdf_y)$.

Hence, $s' \; \mathcal{S} \; \mathfrak{s}'$.

and join Let c be an **and join** connector such that the **and join** rule $s \overset{c}{\Rightarrow} s'$ is applicable for c. Since $s \overset{c}{\Rightarrow} s'$, $\sum_{a \in A_c^{in}} [(a, (i, \bot))] \in m$ for some $i \in \mathbb{N}$ and $m' = m - \sum_{a \in A_c^{in}} [(a, (i, \bot))] + [(a_c^{out}, (i, \bot))]$.

Since $s \; \mathcal{S} \; \mathfrak{s}$, for all $a \in A_c^{in}$, $(a, (i, \bot)) \in m$, there exists $((h(a), i), \iota) \in M$. Then $\mathfrak{s} \overset{c}{\Longrightarrow} \mathfrak{s}'$, where $M' = M - \sum_{a \in A_c^{in}} [((h(a), i), \iota)] + M_c$, $\iota' = \iota$ and

- If $c^\bullet = c' \notin C_{\mathbf{oj}}$, then $M_c = [((h(a_c^{out}), i), \iota)]$. Hence $m'(a_c^{out}, (i, \bot)) = M'((h(a_c^{out}), i), \iota)$ and $(a_c^{out}, (i, \bot)) \; \mathcal{S} \; ((h(a_c^{out}), i), \iota)$.
- If $c^\bullet = y \in C_{\mathbf{oj}}$ and $i \in l$, where $(p_y^u, l) \in M$ and $p_y^u \in h(A_y^{in})$, $M_c = \varnothing$.
- If $a' = (x, y)$ and $y \in C_{\mathbf{oj}}$ and $i \notin l$, where $(p_y^u, l) \in M$ and $p_y^u \in h(A_y^{in})$,
  - If $y \in C_{\mathbf{oj}} \setminus C_t$, $M_c = [((h(A_y^{in}), i :: l), \iota)] - [((h(A_y^{in}), l), \iota)]$.
  - If $y \in C_t$, $M_c = [((p_y^u, i :: l), \iota)] - [((p_y^u, l), \iota)] + [((p_y^t, i), \iota + \iota_y)]$, where $\iota_y \in Dom(pdf_y)$.

In all cases, $s' \; \mathcal{S} \; \mathfrak{s}'$.

xor join Let c be an **xor join** connector, such that the **xor join** rule $s \overset{c}{\Rightarrow} s'$ is applicable for c. Since $s \overset{c}{\Rightarrow} s'$, there exists an incoming arc $a \in A_c^{in}$ so

that $(a,(i,\bot)) \in m$ for some $i \in \mathbb{N}$ and $m' = m - [(a,(i,\bot))] + [(a_c^{out},(i,\bot))]$. Since $s \mathcal{S} \mathfrak{s}$, there exists $((h(a),i),\iota) \in M$.

Then $\mathfrak{s} \overset{c}{\Longrightarrow} \mathfrak{s}'$, where $M' = M - [((h(a),i),\iota)] + M_c$, $\iota' = \iota$ and

- if $c^\bullet \notin C_{\mathbf{oj}}$, $M_c = [((h(a_c^{out}),i),\iota)]$. Hence $m'(a_c^{out},(i,\bot)) = M'((h(a_c^{out}),i),\iota)$ and $(a_c^{out},(i,\bot)) \mathcal{S} ((h(a_c^{out}),i),\iota)$.
- If $c^\bullet = y \in C_{\mathbf{oj}}$ and $i \in l$, where $(p_y^u,l) \in M$ and $p_y^u \in h(A_y^{in})$, $M_c = \varnothing$.
- If $a' = (x,y)$ and $y \in C_{\mathbf{oj}}$ and $i \notin l$, where $(p_y^u,l) \in M$ and $p_y^u \in h(A_y^{in})$,
  - If $y \in C_{\mathbf{oj}} \setminus C_t$, $M_c = [((h(A_y^{in}),i :: l),\iota)] - [((h(A_y^{in}),l),\iota)]$.
  - If $y \in C_t$, $M_c = [((p_y^u,i :: l),\iota)] - [((p_y^u,l),\iota)] + [((p_y^t,i),\iota + \iota_y)]$, where $\iota_y \in Dom(pdf_y)$.

  Hence $s' \mathcal{S} \mathfrak{s}'$.

or join Let c be the **or join** connector such that the **or join** rule $s \overset{c}{\Rightarrow} s'$ is applicable for c.

Since $s \overset{c_f}{\Rightarrow} s'$, $\mathfrak{F} = \sum_{(a,(i,d)) \in m \wedge a \in A_c^{in}} [(a,(i,d))] \neq \varnothing$ for some $i \in \mathbb{N}$ and $d \in \{\bot, 0\}$, and $m' = m + [(a_c^{out},(i,\bot))] - \mathfrak{F}$. Since $s \mathcal{S} \mathfrak{s}$ and $\mathfrak{F}_{\mathfrak{c},\mathfrak{i}} \overset{\text{def}}{=} \sum_{(a,(i,d)) \in m \wedge a \in A_c^{in}} [(a,(i,d))] \neq \varnothing$, we have that $((p_c^t,l),\iota) \in M$ so that $i \in l$ and

- either $c \in C_t$ (has a synchronization timeout), $d = 0$ and $((p_c^t,i),\iota) \in M$, where $p_c^t \in {}^\bullet h(A_c^{in})$, or
- $c \in C_{\mathbf{oj}} \setminus C_t$ and $d = \bot$.

Hence $\mathfrak{s} \overset{c_f}{\Longrightarrow} \mathfrak{s}'$, where $s' = (M',\iota)$ and

- if $c \in C_t$, then $M' = M - [((p_c^u,l),\iota)] - [((p_c^t,i),\iota)] + [((p_c^u,l'),\iota)] + M_c$, where $l' = l \setminus \{i\}$.
- $M' = M - [((p_c^u,l),\iota)] + [((p_c^u,l'),\iota)] + M_c$, where $l' = l \setminus \{i\}$.
  - If $c^\bullet \notin C_{\mathbf{oj}}$, $M_c = [((h(a_c^{out}),i),\iota)]$. Hence $m'(a_c^{out},(i,\bot)) = M'((h(a_c^{out}),i),\iota)$ and $(a_c^{out},(i,\bot)) \mathcal{S} ((h(a_c^{out}),i),\iota)$.
  - If $c^\bullet = y \in C_{\mathbf{oj}}$ and $i \in l$, where $(p_y^u,l) \in M$ and $p_y^u \in h(A_y^{in})$, $M_c = \varnothing$.
  - If $a' = (x,y)$ and $y \in C_{\mathbf{oj}}$ and $i \notin l$, where $(p_y^u,l) \in M$ and $p_y^u \in h(A_y^{in})$,
    - If $y \in C_{\mathbf{oj}} \setminus C_t$, $M_c = [((h(A_y^{in}),l),\iota)] - [((h(A_y^{in}),l \setminus i),\iota)]$.
    - If $y \in C_t$, $M_c = [((p_y^u,i :: l),\iota)] - [((p_y^u,l),\iota)] + [((p_y^t,i),\iota + \iota_y)]$, where $\iota_y \in Dom(pdf_y)$.

  In all cases, $s' \mathcal{S} \mathfrak{s}'$.

time step Let $\mathfrak{t} \in \mathbb{N}^+$ so that $s \overset{\mathfrak{t}}{\Rightarrow} s'$ and let $\mathfrak{F}' = \{\mathfrak{f} \in m | \mathfrak{f}_t > 0\} \neq \varnothing$. Then $\mathfrak{t} = \min\{\mathfrak{f}_t | \mathfrak{f} \in \mathfrak{F}'\} > 0$ and there is no other state $s'' \neq s'$ such that $s \overset{x}{\rightarrow} s''$, where $x \in \Sigma \setminus \mathbb{N}^+$ and $s' = (m', Val)$, where $m' = m + \sum_{(x,(y,\iota'')) \in \mathfrak{F}'} [(x,(y,\iota'' - \mathfrak{t}))] - \mathfrak{F}'$. Since $s \mathcal{S} \mathfrak{s}$, there is no other state $\mathfrak{s}'' \neq \mathfrak{s}'$ such that $\mathfrak{s} \overset{x}{\rightarrow} \mathfrak{s}''$, where $x \in \Sigma' \setminus \mathbb{N}^+$ and $\iota' = \iota + \mathfrak{t}$ is the smallest time at which a binding element (t,b) can become enabled in (M,ι). Then, $(M,\iota) \overset{\mathfrak{t}}{\Longrightarrow} (M,\iota + \mathfrak{t})$. By $s \mathcal{S} \mathfrak{s}$, all the conditions for $s' \mathcal{S} \mathfrak{s}' = (M,\iota + \mathfrak{t})$ to hold are satisfied apart from the condition on the timestamps of the tokens and process folders. For all non-waiting process folders $(x,(y,\iota'')) \in \mathfrak{F}'$, there exists $((p,z),\iota'' + \iota) \in M$ so that $(x,(y,\iota'')) \mathcal{S} ((p,z),\iota'' + \iota)$ and vice versa. Hence, for all non-waiting

process folders $(x, (y, \iota'' - \mathfrak{t})) \in m'$, there exists $((p, z), \iota'' - \mathfrak{t} + \iota + \mathfrak{t}) \in M$ so that $(x, (y, \iota'')) \; \mathcal{S} \; ((p, z), \iota'' + \iota)$, since the global time is $\iota + \mathfrak{t}$ and vice versa. The other conditions can be verified similarly.

($\Leftarrow$) The proof can be done in a similar way. □

Corollary 1.
Let G_e be an eEPC, $U = \langle \mathbb{S}, \Sigma', \Rightarrow, s_0 \rangle$ the transition system of G_e, $\mathcal{N}_e$ its TCPN translation and $U' = \langle \mathfrak{S}(\mathcal{N}_e, \mathfrak{s}_0), \Sigma', \Longrightarrow, \mathfrak{s}_0 \rangle$ the transition system of $\mathcal{N}_e$. The relation $\mathcal{S}$ is a branching bisimulation.

Proof. By Theorem 5.1, $U \leftrightarrows_{\mathsf{w}} U'$ and $\mathcal{S}$ is a weak bisimulation. Since a weak bisimulation where one of the related systems is τ-free is a branching bisimulation [137], it follows that $\mathcal{S}$ is a branching bisimulation. □

Figure 25 shows the translation of the eEPC in Figure 20. Note that as in [47], we can use standard Petri net reduction techniques while preserving properties of the TCPN. Figure 26 shows the translation where instantiations of the function patterns have been reduced to one transition and even merged with an **and join** connector. In general, for functions without any duration which do not operate on data/resources or only perform some operation on attributes, the two steps of pattern can be safely reduced to one.

5.4.2 Verification

A TCPN obtained using the translation procedure described in the previous section can be simulated and analyzed with CPN Tools [75, 37] using state-space analysis (which is basically an exhaustive search through all possible states of the model). We have considered timers that decrease the timeouts and duration, whereas Timed Colored Petri nets consider global time that can increase indefinitely. To deal with the state space explosion caused by the increasing global time, CPN Tools has efficient algorithms that can deal with this source of infinity for bounded nets (e.g. building partial state spaces, the time sweep-line method [31]), which is not always possible in regular model checkers.

The first type of check on eEPCs to be performed is whether the semantics of various types of connectors is respected w.r.t. the ARIS simulation.

If an **or split** has condition events on all its outgoing arcs, at least one of the conditions should be evaluated to *true* at every reachable state which has a process folder on the incoming arc of this connector. Similarly, if an **xor split** has condition events on its outgoing arcs, at most one of the conditions should be evaluated to *true* at every reachable state which has a process folder on the incoming arc of this connector. The violation of such requirements can be easily checked by specifying a marking with at least one token on the place corresponding to the incoming arc of the **or split** such that all boolean expressions on resources or data attributes are *false* as a stop criterion when computing the

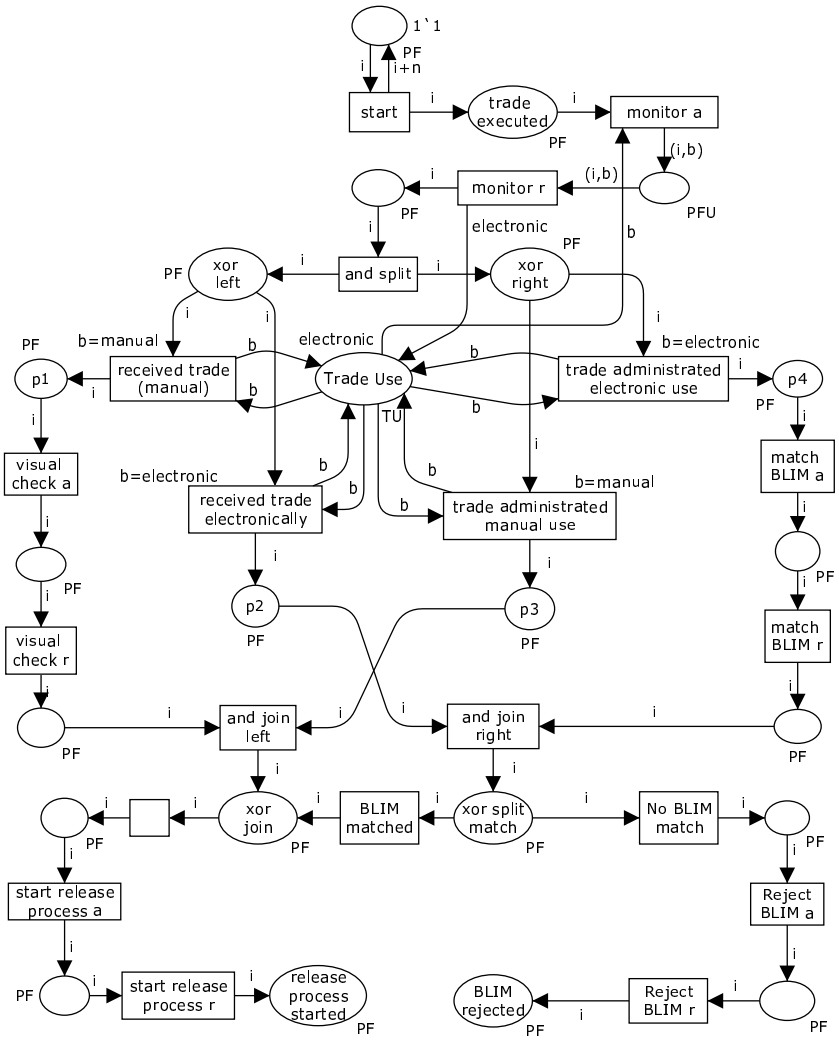


Fig. 25: Translation of the eEPC in Figure 20

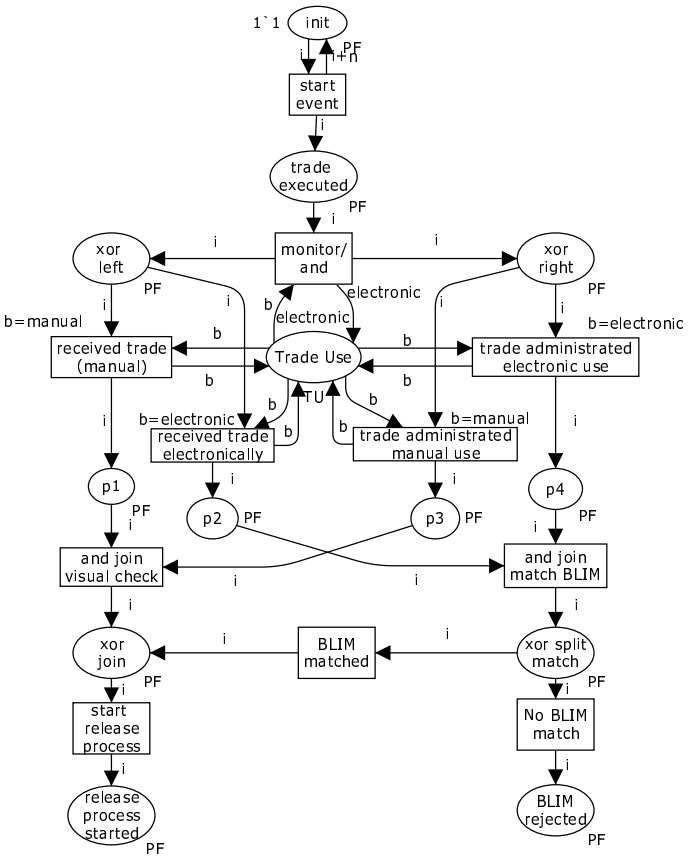


Fig. 26: Reduced TCPN translation of the eEPC in Figure 20

state space. In such case, we can conclude that the eEPC model is not correct and we can provide a simulation of the TCPN that leads to this error.

In case the eEPC does not have timeouts for **or join** connectors and durations for functions, we can analyze the eEPC in which the instantiations of start event set patterns are reduced to places corresponding to the start events in the start event sets. Note that properties that do not hold on the colored net without creation of multiple instances by a same start place corresponding to a start event set, do not hold on the net with creation of multiple instances in the initial places. Hence we can verify the properties directly on the nets without a start event pattern. We may therefore consider as initial markings the marking having a same token on the initial places corresponding to some start event set. In CPN Tools, we can detect whether the eEPC terminates properly, i.e. with all case tokens on places corresponding to outgoing arcs of eEPC nodes adjacent to end events and empty lists on place $h(A_c^{in})$ for each non-timed

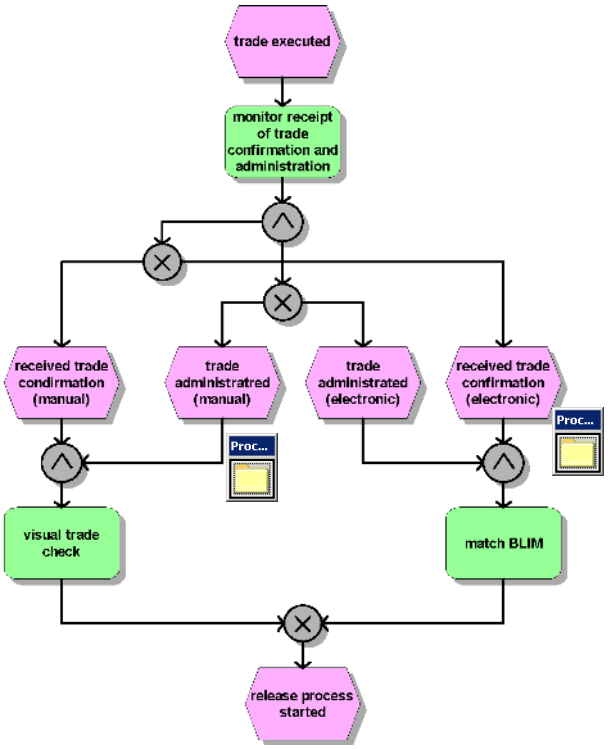
or join connector c. This can be done by investigating the state space and detecting whether the translation has *dead markings* containing only tokens on places corresponding to ingoing arcs of final events, on attribute and resource places and an empty list in place $h(A_c^{in})$ for each non-timed **or join** connector c. For instance, proper termination of the net in Figure 26 (even without multiple instances creation) means that the only dead markings are the ones with exactly one token on exactly one of the final places (*release process started* or *BLIM rejected*).

Below, we give an excerpt of the state space report for Figure 26 without multiple instances creation showing the two dead markings.

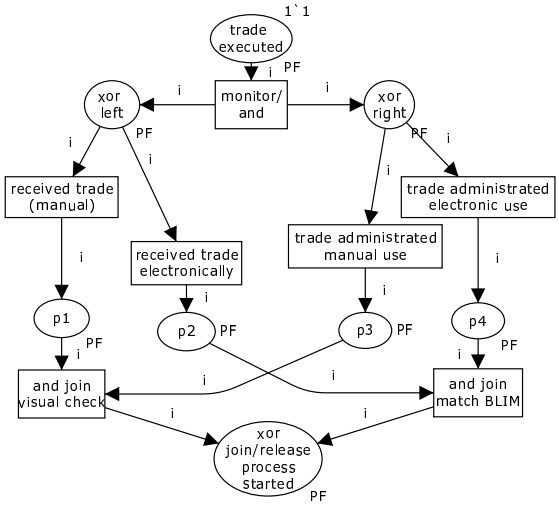
```
Boundedness Properties
----------------------

Best Integer Bounds
                      Upper Lower
t'BLIM_rejected 1       1     0
t'Trade_Use 1           1     1
t'init 1                1     0
t'p1 1                  0     0
t'p2 1                  1     0
t'p3 1                  0     0
t'p4 1                  1     0
t'release_pr_started 1  1     0
trans'trade_executed 1  1     0
trans'xor_join 1        1     0
trans'xor_left 1        1     0
trans'xor_right 1       1     0
trans'xor_split_match 1 1     0

Home Properties

----------------------

Home Markings
Initial Marking is not a
home marking

Liveness Properties
----------------------

Dead Markings
 [8,10]

Dead Transition Instances
 t'and_join_visual_check 1
 t'received_trade 1
 t'trade_admin_manual_use 1

Live Transition Instances
   None
```

Dead markings given by the state space report also provide information about deadlocks in the eEPC, e.g. functions that cannot execute due to non-availability of resources (Figure 28) or non-synchronization, e.g. Figure 27.

```
Boundedness Properties
----------------------

Best Integer Bounds
                      Upper   Lower
dead'p1 1               1       0
dead'p2 1               1       0
dead'p3 1               1       0
dead'p4 1               1       0
dead'trade_executed 1   1       0
dead'xor_join 1         1       0
dead'xor_left 1         1       0
dead'xor_right 1        1       0

Best Upper Multi-set Bounds
     dead'p1 1              1‘1
     dead'p2 1              1‘1
     dead'p3 1              1‘1
     dead'p4 1              1‘1
     dead'trade_executed 1 1‘1
     dead'xor_join 1        1‘1
     dead'xor_left 1        1‘1
```



(a) eEPC



(b) Reduced translation with no generation of new case tokens identifiers

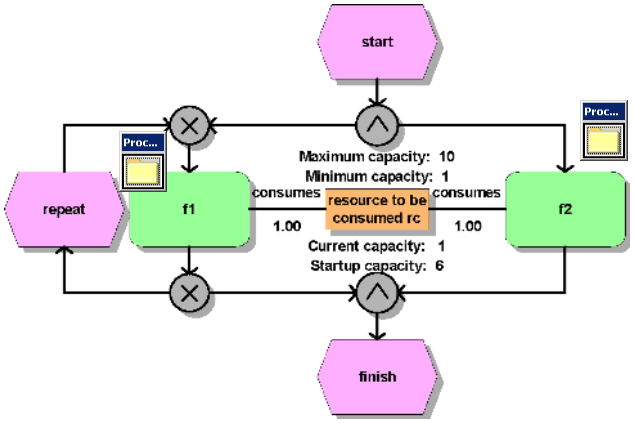
Fig. 27: Deadlock due to non-synchronization of choices

```
     dead'xor_right 1      1`1       Home Markings
                                     Initial Marking is not a
Best Lower Multi-set Bounds          home marking
     dead'p1 1             empty
     dead'p2 1             empty
     dead'p3 1             empty     Liveness Properties
     dead'p4 1             empty     ------------------------
     dead'trade_executed 1 empty
     dead'xor_join 1       empty     Dead Markings
     dead'xor_left 1       empty          [8,9,11]
     dead'xor_right 1      empty
                                     Dead Transition Instances
                                          None
Home Properties
--------------------                 Live Transition Instances
                                          None
```

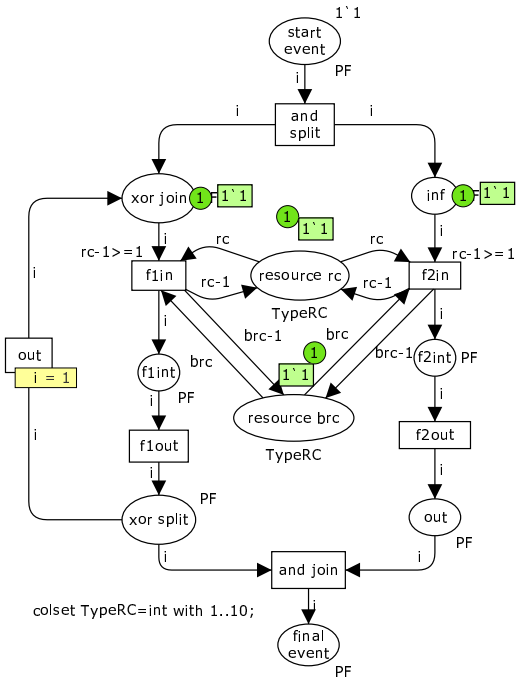
Figure 28 is a typical example of resource starvation, i.e. livelock situations when a function cannot execute since it cannot have access to the required resource. The eEPC shows two function $f1$ and $f2$ which are competing for consuming the same resource (with startup capacity 10 and capacity domain $[1..15]$).

The state space report shows six dead markings. One dead marking representing resource starvation is shown in Figure 28(b) while the corresponding dead state in the eEPC is shown in Figure 28(a).

```
Boundedness Properties               resex'f2int 1       1`1
-----------------------              resex'final_event 1 1`1
                                     resex'inf 1         1`1
Best Integer Bounds                  resex'out 1         1`1
                      Upper  Lower   resex'start_event 1 1`1
resex'resource_brc 1    1      1     resex'xor_join 1    1`1
resex'resource_rc 1     1      1     resex'xor_split 1   1`1
resex'f1int 1           1      0
resex'f2int 1           1      0     Best Lower Multi-set Bounds
resex'final_event 1     1      0     resex'resource_rc 1  empty
resex'inf 1             1      0     resex'resource_brc 1 empty
resex'out 1             1      0     resex'f1int 1        empty
resex'start_event 1     1      0     resex'f2int 1        empty
resex'xor_join 1        1      0     resex'final_event 1  empty
resex'xor_split 1       1      0     resex'inf 1          empty
                                     resex'out 1          empty
 Best Upper Multi-set Bounds         resex'start_event 1  empty
resex'resource_rc 1  1`1++1`2++1`3   resex'xor_join 1     empty
++1`4++1`5++1`6                      resex'xor_split 1    empty
resex'resource_brc 1 1`1++1`2++1`3
++1`4++1`5++1`6
resex'f1int 1       1`1              Home Properties
```



(a) eEPC



(b) Reduced translation with no generation of new case tokens identifiers

Fig. 28: Deadlock due to non-availability of resources

```
-----------------

                                  Dead Markings
Home Markings                     6 [47,45,44,34,24,...]
Initial Marking is not
a home marking                    Dead Transition Instances
                                    None

Liveness Properties               Live Transition Instances
--------------------              None
```

Furthermore, the CPN Tools can verify behavioral properties such as of the model which are defined as temporal logic formulas in ASK-CTL [29].

5.5 Related Work and Conclusions

Related Work There are different approaches to the formalization of the syntax and semantics of EPCs.

One approach is to use Petri nets to specify their semantics. An EPC is translated into a PN using a set of transformation rules. The semantics of EPCs is defined as the semantics of resulting Petri nets. Dehnert [43] and van der Aalst [2] use workflow nets that is suitable to describe EPCs and use specific verification methods developed for PNs in order to verify EPCs. In [2], an EPC is considered to be correct if and only if the workflow obtained as the translation of an EPC is sound. Langner, Schneider and Wehler [86] use a transformation into boolean nets which are colored Petri nets with a single color of type boolean and formulas from the propositional logic as guards. The correctness criterion is the well-formedness of the corresponding boolean net, which is too strict for some practical applications.

Another approach is to consider the transition systems-based semantics. In [107], [7] and [79], the dynamic behavior of an EPC is defined in terms of transition systems. In [7] and [79], the state of an EPC is defined as a mapping of the set of arcs to $\{0, 1\}$ and is represented by the presence or absence of process folders on the arcs of the EPC. Moreover, [7] proposes a non-local semantics of the **xor** and **or** join connector that refers to checking certain conditions that depend on the overall behavior of the EPC. An **xor** join has to wait for a folder on one of its input arcs in order to propagate it to its output arc. However, if a process folder is present or could arrive at some other input arc, the **xor** join should not propagate the folder. For an **or** join, the propagation of a process folder from its input arcs is delayed as long as a process folder could possibly arrive at one of the other input arcs. Computing the transition relation of an EPC has been implemented and tested in [38] using symbolic model checking.

[100] considers also non-local semantics for *or* joins which is given as a transition system. However in [100] the assumption is that the EPCs are safe, i.e. the state space is finite and there is no contact situation. In [47], an EPC is transformed into a workflow net which is checked against soundness and relaxed

soundness. Both approaches consider multiple start/end events and safe EPCs and the users must provide the initial and final states.

In this chapter we considered the semantics of *extended event-driven process chains*, i.e. EPCs extended with data, resources, and time as it is specified in the ARIS Toolset [73]. We provide a formal definition of these semantics in terms of a transition system. Our semantics is instance based which allows us to distinguish specific correctness properties. This can be further used as a base for behavioral (functional) verification of eEPCs using different model checkers.

Furthermore, we provide a translation to timed colored Petri nets and formulate some correctness criteria for eEPCs that can be checked on the translated eEPCs using CPN Tools.

6

Modeling History-dependent Business Processes

Choices in business processes are often based on the process history saved as a log-file listing events and their time stamps. In this chapter we introduce *LogLogics*, a finite-path variant of the Timed Propositional Temporal Logic with Past, which is particularly suitable for specifying guards in business process models. The novelty is due to the presence of boundary points corresponding to the starting and current observation points, which gives rise to a three-valued logic allowing to distinguish between temporal formulas that hold for any log extended with some possible past and future (true), those that do not hold for any extended log (false) and those that hold for some but not all extended logs (unknown). We reduce the check of the truth value of a *LogLogics* formula to a check on a finite abstraction and present an evaluation algorithm. We also define *LogLogics* patterns for commonly occurring properties.

This chapter is based on [69].

6.1 Introduction

An essential feature of workflow management systems (WFMSs) [11] is the distribution of work to agents, which can be either human beings or application software. Decisions taken by a WFMS can depend on previous observations. For instance, a bank can propose more interesting loan conditions to those customers who paid off the previous loans on time. We call processes executed by such a WFMS *history-dependent processes*. Importance of history-based decisions in workflow management has been recognized in the past [120, 122]. In history-dependent processes, actions can be guarded by conditions on the process history. For instance, the **or join** rule with synchronization timeout for eEPCs can be formulated as a timed constraint, allowing the rule to be fired only when certain time has passed from a first enabling of the rule. Adaptive nets support vertical synchronization which can be formulated as constraints on net tokens history, i.e. a transition with an exception guards can fire only if the exception transition is enabled.

Although history-dependent processes are omnipresent in WFMSs, only few models (partially) support them [1, 40]. Since a temporal logic is a natural way to express dependencies between the events observed in the history, those works are based on temporal logics. However, finiteness of the history at any given moment of time and, hence, the inherent incompleteness of observations, should be taken into account, which is not quite adequately done with the classical temporal logics, as their formulas can be evaluated to *true* or *false* only.

To illustrate the resulting limitations consider a guard stating that every bill was paid within four weeks and a log documenting an unpaid bill issued two weeks ago. Following [1, 40] this guard is evaluated to *false*, since there is a bill which is not paid yet. However, the payment term has not expired yet and we do not intend to blacklist the client whom the bill was sent to. Instead we would like to obtain *unknown* in this case, *true* in case every bill was paid and it happened within four weeks, and *false* if there is a bill issued more than four weeks ago that was not paid on time. In some cases the WFMS takes a decision on the continuation of the process giving the benefit of the doubt, i.e. *unknown* leads to the same choice as *true*; in other cases *unknown* leads to the same choice as *false*, and in a number of cases evaluating a guard to *unknown* leads to enabling a special procedure to handle the case.

In this chapter, we propose a new temporal logic, called *LogLogics*, that overcomes the above limitation by reasoning with three truth values. A number of three-valued (untimed) temporal logics, including L-TL, have been proposed by Nakamura [106] and investigated in [102]. Similarly to L-TL, if a *LogLogics*-formula is evaluated to *true* or *false* at a given time point, this value cannot be changed in the future, while *unknown* can become *true* or *false*. Unlike L-TL, not every *LogLogics*-formula has to be eventually evaluated to *true* or *false*.

Since history is a finite linear sequence of timed events, we base *LogLogics* on linear timed temporal logics (defined on infinite sequences) that have been the subject of intensive research in the past, starting with [15, 16, 84]. More recent

works on the subject include [25, 131]. Due to the nature of history, we need to consider not only future but also past temporal operators. Therefore, we have chosen to adapt the Timed Propositional Logic with Past (TPTL+Past) [15, 16].

An alternative to TPTL+Past might have been the metric timed logic (MTL) [84, 131]. The reasons for opting for TPTL+Past rather than for MTL are twofold. First of all, TPTL is "more temporal": it uses real clocks to express timed constraints. This allows to express common WFMS constraints such as "event p occurred between January 1, 2005 and January 1, 2006". Unlike TPTL, MTL reasons in terms of distances between events. Hence, in order to express the same constraint we need to introduce a special event q that occurred on January 1, 2005 and require that p followed q within one year. Second, as recently shown in [25], TPTL+Past is strictly more expressive than MTL+Past.

Two different semantics for timed temporal logics can be considered: point-wise semantics, where formulas are evaluated over discrete sequences of timed events, and interval-based semantics, where formulas are evaluated over the continuous time line [116]. We believe that discrete sequences of timed events, which are actually contained in logs, are better suited for specifying history-based guards in business processes and we choose the point-wise semantics.

We define a *LogLogics*-formula to be *true* for some finite word (log) ρ if it holds for all words containing ρ as a subword, i.e. for the log with all possible pasts and futures. A formula is evaluated to *false* if it does not hold for the log with any of the possible pasts and futures and *unknown* if it holds for the log with some but not all possible pasts and futures. Although defined in terms of infinitely many possible pasts and futures, checking the truth values of a *LogLogics*-formula can be reduced to checking the truth value of the formula on a finite abstraction. We list a number of patterns of commonly occurring guards in business processes and show how these patterns can be expressed in *LogLogics*.

The remainder of the chapter is organized as follows. In Section 6.2 we present *LogLogics*. In Section 6.3 we introduce a finite abstraction that leads us to an evaluation algorithm presented in Section 6.4. In Section 6.5 we show some patterns expressed in our logics. In Section 6.6 we present directions for future research.

6.2 *LogLogics*

In this section we present *LogLogics*, which aims at the modeling of history-dependent processes based on log-files. Log-files record series of *events* such as "100 euro has been withdrawn from account X", "a transaction has failed", "loan Y has been determined to be uncollectible". The set of all events possible in the system is denoted by Σ.

LogLogics is an adaptation of the Next-Free Timed Propositional Temporal Logic with Past [16, 25] to finite sequences of events limited by two special points that refer to the beginning and the end of observations. While the absolute begin is well-suited for modeling the behavior of software systems that have been invoked at some moment of time, it is less appropriate for business processes, where the observations could be available for a recent period of time only. Similarly, there is the last time point where observations are available.

Due to the finiteness of observations, the values of traditional temporal operators can become *unknown*. Consider, for instance, a predicate p that is *true* if a client is reliable. However, the fact that during the entire period of observations the client was reliable does not necessary imply that "always reliable" is true. Nor, in fact does it imply that "always reliable" is false. Indeed, there are two distinct possible futures: one where "always reliable" is true, and another one where "always reliable" is false. In such a situation we would like to say that the value of "always reliable" is *unknown*. To formalize this intuition we start by recapitulating definitions of the well-known Next-Free Timed Propositional Temporal Logics with Past (TPTL+Past) and then define a semantics for finite traces.

We assume that a countable set P of atomic propositions and a countable set V of clock variables are given which are used to assert timed formulas. Then, formulas ϕ are built from atomic propositions, timed formulas, e.g. clock reset $x.\phi$, also known as "freeze", which sets the value of clock x to the current time before evaluating ϕ, comparison of clock values to some nonnegative integer ($x \sim c$) or other clock variable (e.g. $x \leq y + c$), boolean connectives, "until" $\mathcal{U}$ and "since" $\mathcal{S}$ operators, clock constraints and clock resets. Intuitively, $\phi_1 \mathcal{U} \phi_2$ means that at some time point in the future an event happens for which ϕ_2 holds and for all events happened before that event, ϕ_1 holds. Similarly, $\phi_1 \mathcal{S} \phi_2$ means that at some point of time in the past an event happens for which ϕ_2 holds and from that point onwards ϕ_1 holds. Formally:

Definition 6.1. [*LogLogics*]
Formulas ϕ of LogLogics are inductively defined as:

$$\phi := p \mid x \sim y + c \mid x \sim c \mid x.\phi \mid \mathit{false} \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U} \phi_2 \mid \phi_1 \mathcal{S} \phi_2,$$

where $x, y \in V$, $p \in P$, $\sim$ is one of $<, >, \leq, \geq, =, \neq$ and $c \in \mathbb{N}$.

We also assume that the abbreviations $\vee, \Rightarrow, \Leftrightarrow, \mathit{true}$ are defined as usual.

In order to define the formal semantics of *LogLogics* we introduce time sequences and timed words.

Definition 6.2. [Finite/Infinite Timed Words]
A finite time sequence $\tau = \tau_k \tau_{k+1} \ldots \tau_n$ *with $k, n \in \mathbb{Z}$ is a finite sequence of time values $\tau_i \in \mathbb{Z}$, for all $i \in \{k, \ldots, n\}$ such that $\tau_i \leq \tau_{i+1}$ for all $i \in \{k, \cdots, n-1\}$.*

An infinite time sequence $\tau = \tau_k \tau_{k+1} \ldots$ *with $k \in \mathbb{Z}$, is an infinite sequence of times $\tau_i \in \mathbb{Z}$ such that $\tau_i \leq \tau_{i+1}$ for all $i \geq k$.*

A finite event sequence $\sigma = \sigma_k\sigma_{k+1}\ldots\sigma_n$ *with* $k, n \in \mathbb{Z}$ *is a finite sequence of events* $\sigma_i \in \Sigma$, $i \in \{k, \ldots, n\}$. *For any atomic proposition* $p \in P$ *and any* $i \in \{k, \ldots, n\}$, $\sigma_i \vdash p$ *is either true or false.*

An infinite event sequence $\sigma = \sigma_k\sigma_{k+1}\ldots$ *with* $k \in \mathbb{Z}$ *is an infinite sequence of events* $\sigma_i \in \Sigma$, $i \geq k$. *For any atomic proposition* $p \in P$, $\sigma_i \vdash p$, *for* $i \geq k$ *can be evaluated to true or false.*

A finite timed word $\rho = (\sigma, \tau)$ *is a pair consisting of an event sequence* σ *and a time sequence* τ *of the same length. We also write a timed word as a sequence of pairs* $(\sigma_k, \tau_k)\ldots(\sigma_n, \tau_n)$.

An infinite timed word $\rho = (\sigma, \tau)$ *is a pair consisting of an infinite event sequence* σ *and an infinite time sequence* τ.

Note that dates are usual time stamps for business processes (i.e. the exact time is not necessarily indicated in the log), which naturally implies that multiple events can have the same time stamp. Still, also the events with equal time stamps remain ordered and can be in fact causally dependent.

We use the standard semantics of TPTL+Past for infinite traces.

Definition 6.3. [*LogLogics* SEMANTICS ON INFINITE TRACES]
Let ρ *be an infinite timed word. Let* $i \in \mathbb{Z}$ *and* $\nu : V \nrightarrow \mathbb{Z}$ *be a partial valuation for the clock variables. Then*

- $\langle \rho, i, \nu \rangle \vDash p$ *is equal to* $\sigma_i \vdash p$;
- $\langle \rho, i, \nu \rangle \vDash$ *false is false;*
- $\langle \rho, i, \nu \rangle \vDash x \sim c$ *iff* $\nu(x) \sim c$, *where* $\sim \in \{<, >, \leq, \geq, =, \neq\}$ *on* $\mathbb{Z}$, $x \in V$ *and* $c \in \mathbb{N}$;
- $\langle \rho, i, \nu \rangle \vDash x \sim y + c$ *iff* $\nu(x) \sim \nu(y) + c$, *where* $\sim$ *is as above and* $+$ *is the addition on* $\mathbb{Z}$;
- $\langle \rho, i, \nu \rangle \vDash x.\phi$ *iff* $\langle \rho, i, \nu[x \mapsto \tau_i] \rangle \vDash \phi$;
- $\langle \rho, i, \nu \rangle \vDash \neg\phi$ *iff* $\langle \rho, i, \nu \rangle \vDash \phi$ *is false;*
- $\langle \rho, i, \nu \rangle \vDash \phi_1 \wedge \phi_2$ *iff* $\langle \rho, i, \nu \rangle \vDash \phi_1$ *and* $\langle \rho, i, \nu \rangle \vDash \phi_2$;
- $\langle \rho, i, \nu \rangle \vDash \phi_1 \mathcal{U} \phi_2$ *iff* $\langle \rho, j, \nu \rangle \vDash \phi_2$ *for some* $j \geq i$ *and* $\langle \rho, k, \nu \rangle \vDash \phi_1$ *for all* $i \leq k < j$;
- $\langle \rho, i, \nu \rangle \vDash \phi_1 \mathcal{S} \phi_2$ *iff* $\langle \rho, j, \nu \rangle \vDash \phi_2$ *for some* $j \leq i$ *and* $\langle \rho, k, \nu \rangle \vDash \phi_1$ *for all* $j < k \leq i$.

We say that a formula is *closed* if any occurrence of a clock variable x is in the scope of the freeze operator "$x.$". For instance, $x.((x > y + 1) \wedge p)$ is not a closed formula since y does not appear in the scope of "$y.$". One can show in the standard fashion that the truth value of a closed formula is completely defined by the timed word and the time point, i.e., if ϕ is a closed formula, then $\langle \rho, i, \nu \rangle \vDash \phi$ is equivalent to $\langle \rho, i, \beta \rangle \vDash \phi$ for any timed word ρ, time point i and clock valuations ν and β. From here on we restrict our attention to closed formulas.

Based on the temporal operators $\mathcal{S}$ and $\mathcal{U}$ we introduce additional temporal operators "eventually" ($\Diamond\phi := \mathit{true}\mathcal{U}\phi$), "always in the future" ($\Box\phi := \neg(\Diamond\neg\phi)$), "once in the past" ($\Diamondblack\phi := \mathit{true}\mathcal{S}\phi$) and "always in the past" ($\boxminus := \neg(\Diamondblack\neg\phi)$) in

the standard fashion. Observe that $\langle \rho, i, \nu \rangle \vDash \phi$ implies both $\langle \rho, i, \nu \rangle \vDash \Diamond \phi$ and $\langle \rho, i, \nu \rangle \vDash \Diamond\!\!\!\!-\, \phi$. The following proposition provides a more direct way to evaluate formulas using the four additional temporal operators.

Proposition 6.1. *The following statements hold:*

- $\langle \rho, i, \nu \rangle \vDash \Diamond \phi$ *iff* $\langle \rho, j, \nu \rangle \vDash \phi$ *for some* $j \geq i$*;*
- $\langle \rho, i, \nu \rangle \vDash \Box \phi$ *iff* $\langle \rho, j, \nu \rangle \vDash \phi$ *for all* $j \geq i$*;*
- $\langle \rho, i, \nu \rangle \vDash \Diamond\!\!\!\!-\, \phi$ *iff* $\langle \rho, j, \nu \rangle \vDash \phi$ *for some* $j \leq i$*;*
- $\langle \rho, i, \nu \rangle \vDash \boxminus \phi$ *iff* $\langle \rho, j, \nu \rangle \vDash \phi$ *for all* $j \leq i$*.*

Proof. The proof is straightforward from Definition 6.3.

Now we can introduce the semantics of *LogLogics* for *finite timed words*. A log (which is a finite timed word) gives us only partial information about the trace executed by a system (the information about the history before the beginning of the observation can be missing and the information about the future is often not available). Therefore, we will say that a *LogLogics* formula is *true* for a finite timed word $\bar{\rho}$ iff it is true for any infinite timed word obtained by adding to $\bar{\rho}$ a finite prefix and an infinite suffix (i.e. any pre-history prior to τ_k and any future after τ_n). Analogously, a *LogLogics* formula is *false* on a finite timed word if it is false with any pre-history and any future. Finally, we evaluate it to *unknown* if it is neither *true* nor *false* (there is a pre-history and a future that gives us *true* and there is a pre-history and a future that gives us *false*).

Definition 6.4. [Timed Word Extension]
Let $\bar{\rho} = (\bar{\sigma_k}, \bar{\tau_k}) \dots (\bar{\sigma_n}, \bar{\tau_n})$ *be a finite timed word. We define an* extension *of* $\bar{\rho}$ *as an infinite timed word* $\rho = (\sigma_\ell, \tau_\ell)(\sigma_{\ell+1}, \tau_{\ell+1}) \dots$ *satisfying:*

- $\ell \leq k$*;*
- $\bar{\sigma_i} = \sigma_i$ *and* $\bar{\tau_i} = \tau_i$ *for all* $i \in \{k, \dots, n\}$*;*
- *if* $\ell < k$ *then* $\tau_{k-1} < \tau_k$*;*
- $\tau_n < \tau_{n+1}$*.*

Definition 6.5. [*LogLogics* semantics on finite traces]
Let $\bar{\rho} = (\bar{\sigma_k}, \bar{\tau_k}) \dots (\bar{\sigma_n}, \bar{\tau_n})$ *be a finite timed word and* ϕ *a LogLogics-formula. Then*

- $\langle \bar{\rho}, i, \nu \rangle \vDash \phi$ *is true, if for any extension* ρ *of* $\bar{\rho}$*,* $(\langle \rho, i, \nu \rangle \vDash \phi)$ *is true.*
- $\langle \bar{\rho}, i, \nu \rangle \vDash \phi$ *is false, if for any extension* ρ *of* $\bar{\rho}$*,* $(\langle \rho, i, \nu \rangle \vDash \phi)$ *is false.*
- $\langle \bar{\rho}, i, \nu \rangle \nvDash \phi$ *is unknown, if there exist extensions* ρ', ρ'' *of* $\bar{\rho}$ *such that* $(\langle \rho', i, \nu \rangle \nvDash \phi)$ *is true and* $(\langle \rho'', i, \nu \rangle \vDash \phi)$ *is false.*

We abbreviate $\langle \bar{\rho}, n, \epsilon \rangle \vDash \phi$ to $\bar{\rho} \vDash \phi$, where ε is the empty valuation function and n is such that $\bar{\rho} = (\bar{\sigma_k}, \bar{\tau_k}) \dots (\bar{\sigma_n}, \bar{\tau_n})$.

	$\overset{\circ}{\neg}$
false	*true*
unknown	*unknown*
true	*false*

$$(x \overset{\circ}{\wedge} y) \overset{\text{def}}{\Leftrightarrow} \min\{x, y\}$$
$$(\overset{\circ}{\exists} x : x \in S : \pi(x)) \overset{\text{def}}{\Leftrightarrow} \max\{\pi(x) \mid x \in S\}$$
$$(\overset{\circ}{\forall} x : x \in S : \pi(x)) \overset{\text{def}}{\Leftrightarrow} \min\{\pi(x) \mid x \in S\}$$

Fig. 29: Logical connectors and quantifiers in the three-valued logic

6.3 Abstract Timed Words

The difficulty that arises with computing the truth values of a *LogLogics* formula ϕ on a finite timed word $\bar{\rho}$ is that the straightforward application of Definition 6.5 requires in general a check of ϕ on an infinite number of infinite timed words (having $\bar{\rho}$ as a subword). A well-studied approach allowing to reduce the check of a property of an infinite object to a check of a property on a finite approximation of the object is known as *abstraction* [36, 39, 92]. In this section we introduce a notion of an abstract timed word, define a *LogLogics* semantics on abstract timed words and show that the required check can be reduced to a check on the corresponding abstract timed word.

For the sake of brevity, given a set S and a predicate π, we write $\exists x : x \in S : \pi(x)$ and $\forall x : x \in S : \pi(x)$ to denote $\exists x(x \in S \wedge \pi(x))$ and $\forall x(x \in S \Rightarrow \pi(x))$, respectively.

Since by Definition 6.5 the truth value w.r.t. $\bar{\rho}$ can be *true*, *false* or *unknown*, the semantics of a *LogLogics*-formula w.r.t. an abstract timed word should be three-valued as well. Recall that in the traditional three-valued logics (see e.g. [80]) the truth values are ordered as *false* $\prec$ *unknown* $\prec$ *true* and logical connectors and quantifiers are defined in Fig. 29. Note that $\min S$ and $\max S$ are defined for the set S w.r.t. $\prec$. Note that the definitions of $\overset{\circ}{\neg}$, $\overset{\circ}{\exists}$ and $\overset{\circ}{\forall}$ properly extend the corresponding definitions for the two-valued case. In other words, if $\pi(x)$ takes only values *true* or *false* for all $x \in S$ then the truth value of $\overset{\circ}{\exists} x : x \in S : \pi(x)$ in the three-valued logic coincides with its truth value in the two-valued logic, and the same holds for $\overset{\circ}{\forall} x : x \in S : \pi(x)$.

We start with introducing some basic notions we need here.

Abstract time domain

First we extend our time domain to $\mathbb{Z}^{\updownarrow} = \mathbb{Z} \cup \mathbb{Z}^{\uparrow} \cup \mathbb{Z}^{\downarrow}$, $\mathbb{Z}^{\uparrow} = \{\ldots, -1^{\uparrow}, 0^{\uparrow}, 1^{\uparrow}, \ldots\}$ and $\mathbb{Z}^{\downarrow} = \{\ldots, -1^{\downarrow}, 0^{\downarrow}, 1^{\downarrow}, \ldots\}$. Now we pick some arbitrary $a, b, y \in \mathbb{Z}$, $a \leq b$, and define an abstraction function $\alpha_a^b : \mathbb{Z} \to \mathbb{Z}^{\updownarrow}$ and a concretization function

$\gamma : \mathbb{Z}^{\updownarrow} \to 2^{\mathbb{Z}}$ in the following way:

$$\alpha_a^b(x) = \begin{cases} x & \text{if } a \le x \le b \\ b^{\uparrow} & \text{if } x > b \\ a^{\downarrow} & \text{if } x < a \end{cases} \qquad \gamma(z) = \begin{cases} \{z\} & \text{if } z \in \mathbb{Z} \\ \{x \mid x > y\} & \text{if } z = y^{\uparrow} \\ \{x \mid x < y\} & \text{if } z = y^{\downarrow} \end{cases}$$

Following the definition of α and γ, we introduce addition $+_\alpha$ on $\mathbb{Z}^{\updownarrow} \times \mathbb{Z} \to \mathbb{Z}^{\updownarrow}$ and functions $\le_\alpha$ and $=_\alpha$ on $\mathbb{Z}^{\updownarrow} \times \mathbb{Z}^{\updownarrow} \to \{\mathit{false}, \mathit{unknown}, \mathit{true}\}$:

$$z +_\alpha y = \begin{cases} (x+y)^{\uparrow} & \text{if } z = x^{\uparrow}; \\ z + y & \text{if } z \in \mathbb{Z}; \\ (x+y)^{\downarrow} & \text{if } z = x^{\downarrow}; \end{cases}$$

$$(z_1 \le_\alpha z_2) = \begin{cases} \mathit{true} & \text{if } x_1 \le x_2 \text{ for all } x_1 \in \gamma(z_1),\ x_2 \in \gamma(z_2); \\ \mathit{false} & \text{if } x_2 < x_1 \text{ for all } x_1 \in \gamma(z_1),\ x_2 \in \gamma(z_2); \\ \mathit{unknown} & \text{otherwise;} \end{cases}$$

$$(z_1 =_\alpha z_2) = \begin{cases} \mathit{true} & \text{if } z_1, z_2 \in \mathbb{Z} \text{ and } z_1 = z_2; \\ \mathit{false} & \text{if for all } x_i \in \gamma(z_i),\ x_j \in \gamma(z_j),\ x_1 \ne x_2, \\ & \text{for } i, j = 1, 2,\ i \ne j; \\ \mathit{unknown} & \text{otherwise.} \end{cases}$$

Example 1. Consider the expression $7^{\uparrow} +_\alpha 2 \le_\alpha 10^{\downarrow}$.

By the definition of γ, $\gamma(7^{\uparrow}) = \{8, 9, 10, \ldots\}$, i.e. $7^{\uparrow}$ is an abstraction of a whole number which is greater than 7. By the definition of $+_\alpha$, we have that $7^{\uparrow} +_\alpha 2 = (7+2)^{\uparrow} = 9^{\uparrow}$, which coincides with the intuition that tells that by adding 2 to a number greater than 7, we obtain a number greater than 9.

Next we need to compare $9^{\uparrow}$ and $10^{\downarrow}$. By the definition of γ, $\gamma(9^{\uparrow}) = \{10, 11, 12, \ldots\}$ and $\gamma(10^{\downarrow}) = \{\ldots, 7, 8, 9\}$. Clearly, $x \le y$ does not holds for any concretization of $9^{\uparrow}$ and $10^{\downarrow}$. Hence, $7^{\uparrow} +_\alpha 2 \le_\alpha 10^{\downarrow}$ is *false*.

Note that the definitions of $=_\alpha$ and $\le_\alpha$ can be rewritten by case enumeration. So, for the case $x^{\uparrow} \le y^{\uparrow}$ we obtain *unknown* for any x and y since we can always find both concretizations x_c, y_c of $x^{\uparrow}$ and $y^{\uparrow}$ for which the inequality holds, and the ones for which it does not. For the case $x^{\uparrow} \le y$ $(y \in \mathbb{Z})$ we obtain *false* if $x + 1 > y$ and *unknown* otherwise. For $x^{\uparrow} \le y^{\downarrow}$ we obtain *false* if $x + 1 > y - 1$ and *unknown* otherwise; etc.

Example 2. Reconsider $9^{\uparrow} \leq_\alpha 10^{\downarrow}$ from Example 1. Since $9+1 > 10-1$, we obtain *false*.

One can introduce $\geq_\alpha, >_\alpha, <_\alpha, \neq_\alpha$ in the standard way by using $\leq_\alpha$ and $=_\alpha$. We also use $\sim_\alpha$ to refer to an arbitrary abstract comparison.

Abstract timed words

An abstract timed word is a finite word with the special first and last pairs. The first pair is an abstract representation of the period prior to the beginning of the observations while the last pair is an abstract representation of the period after the current moment. These two pairs contain a special event σ^* that does not belong to Σ and denotes an unknown event. For any atomic proposition p we define $\sigma^* \vdash p$ to be *unknown*.

Definition 6.6. [Abstract Timed Words]
An abstract time sequence $\tau^\alpha = \tau_k\tau_{k+1}\dots\tau_n$ *is a finite sequence of times such that* $\tau_k \in \mathbb{Z}^{\downarrow}$, $\tau_n \in \mathbb{Z}^{\uparrow}$, *and for all* i, $k < i < n$, $\tau_i \in \mathbb{Z}$; *moreover,* $\tau_i \leq_\alpha \tau_{i+1}$ *for all* $i \in \{k, \dots, n-1\}$.

An abstract event sequence $\sigma^\alpha = \sigma_k\sigma_{k+1}\dots\sigma_n$ *is a finite sequence of events such that* $\sigma_k = \sigma^*$, $\sigma_n = \sigma^*$, *and for all* i, $k < i < n$, $\sigma_i \in \Sigma$. *For any atomic proposition* $p \in P$, $\sigma_k \vdash p$ *is unknown,* $\sigma_n \vdash p$ *is unknown, and for any* $k < i < n$, $\sigma_i \vdash p$ *is true or false.*

An abstract timed word $\rho^\alpha = (\sigma^\alpha, \tau^\alpha)$ *is a pair consisting of an abstract event sequence* σ^α *and an abstract time sequence* τ^α *of the same length. We also write a timed word as a sequence of pairs* $(\sigma_k, \tau_k)\dots(\sigma_n, \tau_n)$.

Let $\rho = (\sigma, \tau)$ be an infinite timed word with $\tau = \tau_k\tau_{k+1}\dots$ and $\sigma = \sigma_k\sigma_{k+1}\dots$. To relate ρ with an abstract timed word, we extend the abstraction and concretization functions. Let $a \geq k$. Then, the *abstraction of* ρ *w.r.t.* a *and* b is

$$\alpha_a^b(\rho) = (\sigma^*, (\tau_a)^{\downarrow})(\sigma_a, \tau_a)\dots(\sigma_b, \tau_b)(\sigma^*, (\tau_b)^{\uparrow}).$$

The *concretization function* γ maps the abstract timed word $(\sigma^*, (\tau_a)^{\downarrow})$ $(\sigma_a, \tau_a)\dots(\sigma_b, \tau_b)$ $(\sigma^*, (\tau_b)^{\uparrow})$ to the set of all extensions of $(\sigma_a, \tau_a)\dots(\sigma_b, \tau_b)$.

Lemma 6.1. *Let* $\bar\rho = (\bar\sigma_a, \bar\tau_a)\dots(\bar\sigma_b, \bar\tau_b)$ *be a finite timed word. Then for any extension* ρ_1, ρ_2 *of* $\bar\rho$, *we have* $\alpha_a^b(\rho_1) = \alpha_a^b(\rho_2)$.

Proof. Let ρ_1 be $(\sigma^1_\ell, \tau^1_\ell)(\sigma^1_{\ell+1}, \tau^1_{\ell+1})\dots$ and ρ_2 be $(\sigma^2_m, \tau^2_m)(\sigma^2_{m+1}, \tau^2_{m+1})\dots$ so that

- $l \leq a$, $m \leq a$,
- $\bar\tau_i = \tau^1_i = \tau^2_i$ for all $i \in \{a, \dots, b\}$;
- if $l < a$ then $\tau^1_{a-1} < \tau^1_a$, if $m < a$ then $\tau^2_{a-1} < \tau^2_a$;
- $\tau^1_b < \tau^1_{b+1}$ and $\tau^2_b < \tau^2_{b+1}$.

Then, $\alpha_a^b(\rho_1) = (\sigma^*, (\tau^1_a)^{\downarrow})(\sigma^1_a, \tau^1_a)\dots(\sigma^1_b, \tau^1_b)(\sigma^*, (\tau^1_b)^{\uparrow}) = \alpha_a^b(\rho_2)$. □

Next we define the semantics of a *LogLogics*-formula w.r.t. an abstract timed word and an abstract clock valuation.

Definition 6.7. [*LogLogics* Semantics on Abstract Traces]
Let $\rho^\alpha = (\sigma_{a-1}, (\tau_a)^\downarrow)(\sigma_a, \tau_a)\dots(\sigma_b, \tau_b)(\sigma_{b+1}, (\tau_b)^\uparrow)$ *be an abstract timed word, where* $\sigma_{a-1} = \sigma_{b-1} = \sigma^*$, $i \in \mathbb{Z}$, $a-1 \le i \le b+1$, *and* $\nu^\alpha : V \nrightarrow \mathbb{Z}^\updownarrow$ *be a partial valuation for the clock variables. Then*

- $\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash p$ *is equal to* $\sigma_i \vdash p$;
- $\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash$ *false equals false;*
- $\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash x \sim c$ *is equal to the value of* $\nu^\alpha(x) \sim_\alpha c$, *where* $\sim_\alpha$ *is the relation on* $\mathbb{Z}^\updownarrow$ *corresponding to* $\sim$ *and* $c \in \mathbb{N}$;
- $\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash x \sim y + c$ *is equal to the value of* $\nu^\alpha(x) \sim_\alpha \nu^\alpha(y) +_\alpha c$, *where* $\sim_\alpha$ *is as above and* $c \in \mathbb{N}$;
- $\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash x.\phi$ *is equivalent to* $\langle \rho^\alpha, i, \nu^\alpha[x \mapsto \tau_i] \rangle \vDash \phi$;
- $\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash \neg\phi$ *is equivalent to* $\mathring{\neg}(\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash \phi)$;
- $\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash \phi_1 \wedge \phi_2$ *is equivalent to* $(\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash \phi_1) \mathring{\wedge} (\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash \phi_2)$
- $\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash \phi_1 \mathcal{U} \phi_2$ *is equivalent to* $\mathring{\exists} j : i \le j : (\langle \rho^\alpha, j, \nu^\alpha \rangle \vDash \phi_2 \mathring{\wedge} \mathring{\forall} k : i \le k < j : \langle \rho^\alpha, k, \nu^\alpha \rangle \vDash \phi_1)$;
- $\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash \phi_1 \mathcal{S} \phi_2$ *is equivalent to* $\mathring{\exists} j : j \le i : (\langle \rho^\alpha, j, \nu^\alpha \rangle \vDash \phi_2 \mathring{\wedge} \mathring{\forall} k : j < k \le i : \langle \rho^\alpha, k, \nu^\alpha \rangle \vDash \phi_1)$.

We also abbreviate $\langle \rho^\alpha, b, \varepsilon \rangle \vDash \phi$ to $\rho^\alpha \vDash \phi$.

The set of subformulae $\mathrm{Sub}(\phi)$ of a *LogLogics* formula ϕ is defined inductively as follows:

$$
\begin{aligned}
\mathrm{Sub}(p) &:= \{p\};\\
\mathrm{Sub}(x \sim y + c) &:= \{x \sim y + c\};\\
\mathrm{Sub}(x \sim c) &:= \{x \sim c\};\\
\mathrm{Sub}(x.\phi) &:= \mathrm{Sub}(\phi) \cup \{x.\phi\};\\
\mathrm{Sub}(\mathit{false}) &:= \{\mathit{false}\};\\
\mathrm{Sub}(\neg\phi) &:= \mathrm{Sub}(\phi) \cup \{\neg\phi\};\\
\mathrm{Sub}(\phi' \wedge \phi'') &:= \mathrm{Sub}(\phi') \cup \mathrm{Sub}(\phi'') \cup \{\phi' \wedge \phi''\};\\
\mathrm{Sub}(\phi' \mathcal{U} \phi'') &:= \mathrm{Sub}(\phi') \cup \mathrm{Sub}(\phi'') \cup \{\phi' \mathcal{U} \phi''\};\\
\mathrm{Sub}(\phi' \mathcal{S} \phi'') &:= \mathrm{Sub}(\phi') \cup \mathrm{Sub}(\phi'') \cup \{\phi' \mathcal{S} \phi''\},
\end{aligned}
$$

where $x, y \in V$, $p \in P$, $\sim \in \{<, >, \le, \ge, =, \ne\}$, $c \in \mathbb{N}$ and ϕ', ϕ'' are *LogLogics*-formulae.

A *proper subformula* of a *LogLogics* formula ϕ is a formula from $\mathrm{Sub}(\phi) \setminus \{\phi\}$.

Theorem 6.1. *Let* $\bar\rho = (\bar\sigma_k, \bar\tau_k)\dots(\bar\sigma_n, \bar\tau_n)$ *be a finite timed word,* $\mathcal{P}$ *the set of all extensions of* $\bar\rho$, $\rho^\alpha = (\sigma^*, \bar\tau_k{}^\downarrow)\bar\rho(\sigma^*, \bar\tau_n{}^\uparrow)$ *an abstract word,* ϕ *a LogLogics-formula,* $i \in \mathbb{Z}$ *such that* $k-1 \le i \le n+1$, $\nu^\alpha : V \nrightarrow \mathbb{Z}^\updownarrow$ *a partial abstract clock valuation and* $\mathfrak{V} = \{\nu : V \nrightarrow \mathbb{Z} \mid \forall x : x \in V : \nu(x) \in \gamma(\nu^\alpha(x)) \wedge \alpha^{\bar\tau_n}_{\bar\tau_k}(\nu(x)) = \nu^\alpha(x)\}$ *the set of concrete clock valuations corresponding to the abstract clock valuation* ν^α. *Then the truth value of* $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi)$ *equals the truth value of* $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \phi$.

Proof. We proceed by induction on the structure of ϕ assuming that the theorem holds for all proper subformulae of ϕ.

We only prove that $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi$ is *true* iff $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash \rho)$ is *true*. The proofs for *false* and *unknown* are similar.
($\Rightarrow$) Let $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash \phi)$ be *true*. We show that $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi$ is *true*.

Let $x, y \in V$, $p \in P$, $\sim \in \{\leq, =\}$ (the rest of the cases can be derived), $c \in \mathbb{N}$ and ϕ_1, ϕ_2 be *LogLogics*-formulae.

$\phi :=$ p Since $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash p)$ is *true*, then $\sigma_i \vdash p$, hence $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash p$ is *true*.

$\phi :=$ *false* Since $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash \mathit{false})$ is *false* for all $\rho \in \mathcal{P}$ and $\nu \in \mathfrak{V}$, this case is not applicable.

$\phi :=$ $x \sim c$ Since $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash x \sim c)$ is *true*, $\nu(x) \sim c$ holds, for all $\rho \in \mathcal{P}$ and $\nu \in \mathfrak{V}$. We show that $\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash x \sim c$ is *true*, i.e. $\nu^\alpha(x) \sim_\alpha c$ holds for $\sim \in \{\leq, =\}$.

If **$\phi :=$ $\nu(x) = c$**, we have the following cases:

- **$\nu^\alpha(x) = \bar{\tau_n}^\uparrow$** is not applicable. If it were applicable, then $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash \phi)$ would be *true*, however, we can take a $\nu \in \mathfrak{V}$ with $\bar{\tau_n} < \nu(x) \neq c$, which forms a contradiction.
- **$\nu^\alpha(x) = \bar{\tau_k}^\downarrow$** is not applicable. If it were applicable, then $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash \phi)$ would be *true*; however, we can take a $\nu \in \mathfrak{V}$ with $\bar{\tau_k} < \nu(x) \neq c$, which forms a contradiction.
- **$\bar{\tau_k} \leq \nu^\alpha(x) \leq \bar{\tau_n}$** Then $\nu^\alpha(x) = \nu(x)$ and since $\nu(x) = c$ is *true* for all $\nu \in \mathfrak{V}$, we have that $\nu^\alpha(x) =_\alpha c$ is *true*, hence $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi$ is *true*.

If **$\phi :=$ $\nu(x) \leq c$**, we have the following cases:

- **$\nu^\alpha(x) = \bar{\tau_n}^\uparrow$** is not applicable. If it were applicable, then $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash \phi)$ would be *true*; however, we can take a $\nu \in \mathfrak{V}$ with $\bar{\tau_n} < \nu(x)$ and $\nu(x) > c$, which forms a contradiction.
- **$\nu^\alpha(x) = \bar{\tau_k}^\downarrow$** is not applicable. If it were applicable, then $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash \phi)$ would be *true*; however, we can take a $\nu \in \mathfrak{V}$ with $\bar{\tau_k} < \nu(x)$ and $\nu(x) > c$, which forms a contradiction.
- **$\bar{\tau_k} \leq \nu^\alpha(x) \leq \bar{\tau_n}$** Then $\nu^\alpha(x) = \nu(x)$ and since $\nu(x) \leq c$, we have $\nu^\alpha(x) \leq_\alpha c$.

$\phi :=$ $x \sim y + c$ Then $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash x \sim y + c)$ is *true* and we shall show that $\nu^\alpha(x) \sim_\alpha \nu^\alpha(y) +_\alpha c$ holds, i.e. $\langle \alpha_k^n(\rho), i, \nu \rangle \vDash x \sim y + c$ is *true*.

If **$\phi :=$ $x = y + c$**, we have the following cases:

- **$\nu^\alpha(x) = \bar{\tau_n}^\uparrow$** is not applicable. If it were applicable, then $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash \phi)$ is *true*, however, we can take a $\nu \in \mathfrak{V}$ with $\bar{\tau_n} < \nu(x) \neq \nu(y) + c$, which forms a contradiction.
- **$\nu^\alpha(x) = \bar{\tau_k}^\downarrow$** is not applicable. If it were applicable, then $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash \phi)$ would be *true*; however, we can take a $\nu \in \mathfrak{V}$ with $\bar{\tau_k} > \nu(x) \neq \nu(y) + c$, which forms a contradiction.
- **$\bar{\tau_k} \leq \nu^\alpha(x) \leq \bar{\tau_n}$**

- $\bar{\tau_k} \leq \nu^\alpha(y) \leq \bar{\tau_n}$ Then for all $\rho \in \mathcal{P}$ and $\nu \in \mathfrak{V}$, $\langle \rho, i, \nu \rangle \vDash \nu(x) = \nu(y) + c$ is *true*. Since $\nu^\alpha(x) = \nu(x)$ and $\nu^\alpha(y) = \nu(y)$, we have that $\nu^\alpha =_\alpha \nu^\alpha(y) +_\alpha c$. Hence $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi$ is *true*.
- $\nu^\alpha(y) = \bar{\tau_k}^\downarrow$ is not applicable. If it were applicable, then $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi)$ would be *true*; however, we can take a $\nu \in \mathfrak{V}$ with $\bar{\tau_k} > \nu(y) + c \neq \nu(x)$, which forms a contradiction.
- $\nu^\alpha(y) = \bar{\tau_n}^\uparrow$ is not applicable. If it were applicable, then $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi)$ is *true*; however, we can take a $\nu \in \mathfrak{V}$ with $\nu(x) \neq \nu(y) + c > \bar{\tau_n}$, which forms a contradiction.

If $\phi := x \leq y + c$, we have the following cases:

- $\nu^\alpha(x) = \bar{\tau_n}^\uparrow$ is not applicable. If it were applicable, then $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi)$ would be *true*, however, we can take a $\nu \in \mathfrak{V}$ with $\bar{\tau_n} < \nu(x)$ and $\nu(x) > \nu(y) + c$, which forms a contradiction.
- $\nu^\alpha(x) = \bar{\tau_k}^\downarrow$
  - $\nu^\alpha(y) = \bar{\tau_n}^\uparrow$ Then $\nu^\alpha(y) +_\alpha c = (\bar{\tau_n} + c)^\uparrow$, and since $\nu(x) \leq \nu(y) + c$, we have that $\nu^\alpha(x) = \bar{\tau_k}^\downarrow \leq_\alpha \nu^\alpha(y) +_\alpha c$.
  - $\nu^\alpha(y) = \bar{\tau_k}^\downarrow$ is not applicable. If it were applicable, then $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi)$ would be *true*; however, we can take a $\nu \in \mathfrak{V}$ with $\bar{\tau_k} > \nu(x) > \nu(y) + c$, which forms a contradiction.
  - $\bar{\tau_k} \leq \nu^\alpha(y) \leq \bar{\tau_n}$ Then $\nu^\alpha(y) +_\alpha c = \nu(y) + c$, and since $\nu(x) \leq \nu(y) + c$, we have that $\nu^\alpha(x) = \bar{\tau_k}^\downarrow \leq_\alpha \nu^\alpha(y) +_\alpha c$.
- $\bar{\tau_k} \leq \nu^\alpha(x) \leq \bar{\tau_n}$
  - $\nu^\alpha(y) = \bar{\tau_n}^\uparrow$ Then $\nu^\alpha(x) = \nu(x) \leq \nu(y) + c$ and $\nu^\alpha(y) +_\alpha c = \bar{\tau_n} + c^\uparrow$, therefore $\nu^\alpha(x) \leq_\alpha \nu^\alpha(y) +_\alpha c$.
  - $\nu^\alpha(y) = \bar{\tau_k}^\downarrow$ is not applicable since we know that $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi)$ is *true*, however, we can take a $\nu \in \mathfrak{V}$ with $\bar{\tau_k} > \nu(x) > \nu(y) + c$, which forms a contradiction.
  - $\bar{\tau_k} \leq \nu^\alpha(y) \leq \bar{\tau_n}$ Since $\nu(x) \leq \nu(y) + c$, $\nu^\alpha(y) = \nu(y)$ and $\nu^\alpha(x) = \nu(x)$, therefore $\nu^\alpha(x) \leq_\alpha \nu^\alpha(y) +_\alpha c$.

$\phi := x.\phi_1$ Since $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash x.\phi_1)$ is *true*, then $\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu[x \mapsto \tau_i] \rangle \vDash \phi_1$ is *true*. Let $\mathfrak{V}' = \{\nu' : V \nrightarrow \mathbb{Z} | \forall x : x \in V : \nu'(x) \in \gamma(\nu^\alpha[x \mapsto \alpha_{\bar{\tau_k}}^{\bar{\tau_n}}(\tau_i)]) \wedge \alpha_{\bar{\tau_k}}^{\bar{\tau_n}}(\nu'(x)) = \nu^\alpha[x \mapsto \alpha_{\bar{\tau_k}}^{\bar{\tau_n}}(\tau_i)]\}$ be the set of concrete clock valuations corresponding to the abstract clock valuation $(\nu^\alpha[x \mapsto \alpha_{\bar{\tau_k}}^{\bar{\tau_n}}(\tau_i)]$. Then $\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V}' : \langle \rho, i, \nu[x \mapsto \tau_i] \rangle \vDash \phi_1$ is *true*.
By the induction hypothesis, we have that $\langle \alpha_k^n(\rho), i, \nu^\alpha[x \mapsto \alpha_{\bar{\tau_k}}^{\bar{\tau_n}}(\tau_i)] \rangle \vDash \phi_1$ is *true*, hence $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash x.\phi_1$ is *true*.

$\phi := \neg\phi_1$ $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \neg\phi_1)$ is *true* iff $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi_1)$ is *false*. By the induction hypothesis $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi_1$ is *false*. Hence $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \neg\phi_1$ is *true*.

$\phi := \phi_1 \wedge \phi_2$ $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi_1 \wedge \phi_2)$ holds iff $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi_1)$ and $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi_2)$ hold. By the induction hypothesis $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi_1$ and $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi_2$ hold. Hence $\min\{\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi_1, \langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi_2\} = true$, thus $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi_1 \wedge \phi_2$.

$\boldsymbol{\phi := \phi_1 \mathcal{U} \phi_2}$ Since $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash \phi_1 \mathcal{U} \phi_2)$ holds, we have that $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, j, \nu \rangle \vDash \phi_2)$ is *true* for some $j \geq i$ and $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, k', \nu \rangle \vDash \phi_1)$ is *true* for all k' satisfying $i \leq k' < j$.
We only look at the case when $j \leq n$, since for $j > n$, the only case when $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, j, \nu \rangle \vDash \phi_2)$ is *true* is when $\phi_1 \equiv$ *true*, and in this case $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi_1 \mathcal{U} \phi_2$ is trivially *true*.
By the induction hypothesis, we have that $\langle \alpha_n^k(\rho), j, \nu^\alpha \rangle \vDash \phi_2$ is *true* for some $n \geq j \geq i$ and $\langle \alpha_n^k(\rho), k', \nu^\alpha \rangle \vDash \phi_1$ is *true* for all k' satisfying $i \leq k' < j$. Hence $\max\{\langle \alpha_k^n(\rho), j, \nu^\alpha \rangle \vDash \phi_2 | j \geq i\}$ is *true*, therefore $\mathring{\exists} j : j \leq i : \langle \alpha_k^n(\rho), j, \nu^\alpha \rangle \vDash \phi_2$ is *true*. Similarly, $\min\{\langle \alpha_k^n(\rho), k', \nu^\alpha \rangle \vDash \phi_1 | i \leq k' < j\}$ is *true*, hence $\mathring{\forall} k' : i < k' \leq j : \langle \alpha_k^n(\rho), k, \nu^\alpha \rangle \vDash \phi_1$ is *true*. Hence, $\min\{\mathring{\exists} j : j \leq i : \langle \alpha_k^n(\rho), j, \nu^\alpha \rangle \vDash \phi_2, \mathring{\forall} k' : n \leq k' < j : \langle \alpha_k^n(\rho), k', \nu^\alpha \rangle \vDash \phi_1\}$ is *true* thus $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi_1 \mathcal{U} \phi_2$.

$\boldsymbol{\phi := \phi_1 \mathcal{S} \phi_2}$ $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash \phi_1 \mathcal{S} \phi_2)$ is *true* iff $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, j, \nu \rangle \vDash \phi_2)$ is *true* for some $j \leq i$ and $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, k', \nu \rangle \vDash \phi_1)$ is *true* for all $j < k' \leq i$.
We only look at the case when $k \leq j$, since for $k > j$, the only case when $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, j, \nu \rangle \vDash \phi_2)$ is *true* is when $\phi_2 \equiv$ *true*, and in this case $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi_1 \mathcal{S} \phi_2$ is trivially *true*.
By the induction hypothesis $\langle \alpha_k^n(\rho), j, \nu^\alpha \rangle \vDash \phi_1$ is *true* for some $j \leq i$ and $\langle \alpha_k^n(\rho), k', \nu^\alpha \rangle \vDash \phi_2$ holds for all k' satisfying $j < k' \leq i$.
Hence $\max\{\langle \alpha_k^n(\rho), j, \nu^\alpha \rangle \vDash \phi_2 | k \leq j \leq i\}$ is *true* and $\min\{\langle \alpha_k^n(\rho), k', \nu^\alpha \rangle \vDash \phi_1 | j < k' \leq i\}$ is *true*. Thus $\min\{\mathring{\exists} j : j \leq n : \langle \alpha_k^n(\rho), j, \nu^\alpha \rangle \vDash \phi_2, \mathring{\forall} k : j < k \leq i : \langle \alpha_k^n(\rho), k', \nu^\alpha \rangle \vDash \phi_1\}$ is *true*, thus $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi_1 \mathcal{S} \phi_2$.

($\Leftarrow$) Let $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi$ be *true*. We show that $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash \phi)$ is *true*.

Let $x, y \in V$, $p \in P$, $\sim \in \{\leq, =\}$ (the rest of the cases can be derived), $c \in \mathbb{N}$ and ϕ_1, ϕ_2 be *LogLogics*-formulae.

$\boldsymbol{\phi := p}$ Since $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash p$ is *true*, then $\sigma_i \vdash p$, hence $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash p)$ is *true*.

$\boldsymbol{\phi :=}$ ***false*** $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash$ *false* is *false*, which is not applicable in this case.

$\boldsymbol{\phi := x \sim c}$ Since $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash x \sim c$ is *true*, $\nu^\alpha(x) \sim_\alpha c$ holds. We show that $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash x \sim c)$ is *true* for $\sim \in \{\leq, =\}$. Let $\rho \in \mathcal{P}$ be an arbitrary extension of $\bar{\rho}$ and $\nu \in \mathfrak{V}$ an arbitrary clock valuation.
If $\boldsymbol{\phi := x = c}$ we have the following cases:
- $\boldsymbol{\bar{\tau}_k \leq \nu(x) \leq \bar{\tau}_n}$ Since $\nu^\alpha(x) = \nu(x)$ and $\nu^\alpha(x) =_\alpha c$, we have that $\nu(x) = c$. Then $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash x = c)$ is *true*.
- $\boldsymbol{\bar{\tau}_n < \nu(x)}$ is not applicable, since $\alpha^{\bar{\tau}_n}_{\bar{\tau}_n}(\nu(x)) = \bar{\tau}_n{}^{\uparrow}$ and the truth value of $\bar{\tau}_n{}^{\uparrow} =_\alpha c$ is *unknown*, while $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi$ is *true*.
- $\boldsymbol{\bar{\tau}_k > \nu(x)}$ is not applicable, since $\alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu(x)) = \bar{\tau}_k{}^{\downarrow}$ and the truth value of $\bar{\tau}_k{}^{\downarrow} =_\alpha c$ is *unknown*, while $\langle \alpha_k^n(\rho), i, \nu^\alpha \rangle \vDash \phi$ is *true*.

If $\boldsymbol{\phi := x \leq c}$ we have the following cases:
- $\boldsymbol{\bar{\tau}_k \leq \nu(x) \leq \bar{\tau}_n}$ Since $\nu^\alpha(x) = \nu(x)$ and $\nu^\alpha(x) \leq_\alpha c$, we have that $\nu(x) \leq c$. Then $(\forall \rho, \nu \colon \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} \colon \langle \rho, i, \nu \rangle \vDash x \leq c)$ is *true*.

- $\bar{\tau}_n < \nu(x)$ is not applicable, since $\alpha^{\bar{\tau}_n}_{\bar{\tau}_n}(\nu(x)) = \bar{\tau}_n{}^{\uparrow}$ and the truth value of $\bar{\tau}_n{}^{\uparrow} \leq_\alpha c$ is *unknown*, while $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \models \phi$ is *true*.
- $\bar{\tau}_k > \nu(x)$ is not applicable, since $\alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu(x)) = \bar{\tau}_k{}^{\downarrow}$ and the truth value of $\bar{\tau}_k{}^{\downarrow} \leq_\alpha c$ is *unknown*, while $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \models \phi$ is *true*.

$\phi := x \sim y + c$ Then $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \models x \sim y + c$ is *true* and we shall show that $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \models x \sim y + c)$ is *true*. Let $\rho \in \mathcal{P}$ be an arbitrary extension of $\bar{\rho}$ and $\nu \in \mathfrak{V}$ an arbitrary clock valuation.

If $\phi := x = y + c$, we have the following cases:

- $\bar{\tau}_k \leq \nu(x) \leq \bar{\tau}_n$ Then $\nu^\alpha(x) = \nu(x)$.
  - $\bar{\tau}_k \leq \nu(y) \leq \bar{\tau}_n$ Then $\nu^\alpha(y) +_\alpha c = \nu(y) + c$ and since $\nu^\alpha(x) =_\alpha \nu^\alpha(y) +_\alpha c$ and $\nu^\alpha(x) = \nu(x)$, we have $\nu(x) = \nu(y) + c$ for all $\nu \in \mathfrak{V}$. Hence $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \models \phi)$ is *true*.
  - $\bar{\tau}_k < \nu(y)$ is not applicable, since $\alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu(y)) +_\alpha c = (\bar{\tau}_k + c)^{\downarrow}$ and the truth value of $\nu(x) =_\alpha (\bar{\tau}_k + c)^{\downarrow}$ is *unknown*, while $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \models \phi$ is *true*.
  - $\bar{\tau}_n > \nu(y)$ is not applicable, since $\alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu(y)) +_\alpha c = (\bar{\tau}_n + c)^{\uparrow}$ and the truth value of $\nu(x) =_\alpha (\bar{\tau}_n + c)^{\uparrow}$ is *false*, while $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \models \phi$ is *true*.
- $\bar{\tau}_k > \nu(x)$ is not applicable, since $\alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu(x)) = \bar{\tau}_k{}^{\downarrow}$ and the truth value of $\bar{\tau}_k{}^{\downarrow} =_\alpha \alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu(y)) +_\alpha c$ is *unknown*, while $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \models \phi$ is *true*.
- $\bar{\tau}_n < \nu(x)$ is not applicable, since $\alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu(x)) = \bar{\tau}_n{}^{\uparrow}$ and the truth value of $\bar{\tau}_k{}^{\downarrow} =_\alpha \alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu(y)) +_\alpha c$ is either *false* or *unknown*, while $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \models \phi$ is *true*.

If $\phi := x \leq y + c$ we have the following cases:

- $\bar{\tau}_k \leq \nu(x) \leq \bar{\tau}_n$ Then $\nu^\alpha(x) = \nu(x)$.
  - $\bar{\tau}_k \leq \nu(y) \leq \bar{\tau}_n$ Then $\nu^\alpha(y) +_\alpha c = \nu(y) + c$ and since $\nu^\alpha(x) \leq_\alpha \nu^\alpha(y) +_\alpha c$, we have that for all $\nu \in \mathfrak{V}$, $\nu(x) \leq \nu(y) + c$. Hence $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \models x \leq y + c)$ is *true*.
  - $\nu(y) > \bar{\tau}_n$ Then $\nu^\alpha(y) +_\alpha c = \bar{\tau}_n{}^{\uparrow} + c$, and since $\nu^\alpha(x) \leq_\alpha \nu^\alpha(y) +_\alpha c$ is *true*, we have that for all $\nu \in \mathfrak{V}$, $\nu(x) \leq \nu(y) + c$. Hence $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \models x \leq y + c)$ is *true*.
  - $\bar{\tau}_k > \nu(y)$ is not applicable since $\nu(x) \leq_\alpha \alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu(y)) +_\alpha c = (\bar{\tau}_k + c)^{\downarrow}$ is *unknown*, while the induction hypothesis is that $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \models \phi$ is *true*.
- $\bar{\tau}_k > \nu(x)$ Then $\nu^\alpha(x) = \bar{\tau}_k{}^{\downarrow}$.
  - $\bar{\tau}_k \leq \nu(y) \leq \bar{\tau}_n$ We have that $\nu^\alpha(x) = \bar{\tau}_k{}^{\downarrow} \leq_\alpha \nu^\alpha(y) +_\alpha c = \nu(y) + c$ is *true*. Clearly for all $\nu \in \mathfrak{V}$ satisfying the above properties, we have $\nu(x) \leq \nu(y) + c$, hence $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \models x \leq y + c)$ is *true*.
  - $\bar{\tau}_k > \nu(y)$ is not applicable since $\bar{\tau}_k{}^{\downarrow} \leq_\alpha \alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu(y)) +_\alpha c = (\bar{\tau}_k + c)^{\downarrow}$ is *unknown*, while our hypothesis is that $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \models \phi$ is *true*.
  - $\bar{\tau}_n < \nu(y)$ Then $\nu^\alpha(x) = \bar{\tau}_k{}^{\downarrow} \leq_\alpha \nu^\alpha(y) +_\alpha c = (\bar{\tau}_n + c)^{\uparrow}$ is *true*. Clearly, for all $\nu \in \mathfrak{V}$, $\rho \in \mathcal{P}$ satisfying the above properties, we have $\nu(x) \leq \nu(y) + c$, hence $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \models x \leq y + c)$ is *true*.

– $\boldsymbol{\bar{\tau}_n < \nu(x)}$ Then $\alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu(x)) = \bar{\tau}_n{}^{\uparrow}$.
  - $\boldsymbol{\nu(y) > \bar{\tau}_n}$ is not applicable since $\bar{\tau}_n{}^{\uparrow} \leq_\alpha \alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu(y)) +_\alpha c = (\bar{\tau}_n + c)^{\uparrow}$ is *unknown* while our hypothesis is that $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \phi$ is *true*.
  - $\boldsymbol{\bar{\tau}_k > \nu(y)}$ is not applicable since $\bar{\tau}_n{}^{\uparrow} \leq_\alpha \alpha^{\bar{\tau}_k}_{\bar{\tau}_k}(\nu(y)) +_\alpha c = (\bar{\tau}_k + c)^{\downarrow}$ is *false* when $\bar{\tau}_n + 1 \leq \bar{\tau}_k + c - 1$ and *unknown* otherwise, while our hypothesis is that $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \phi$ is *true*.
  - $\boldsymbol{\bar{\tau}_k \leq \nu(y) \leq \bar{\tau}_n}$ is not applicable since $\bar{\tau}_n{}^{\uparrow} \leq_\alpha \alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu(y)) +_\alpha c = \nu(y) + c$ is *false* when $\bar{\tau}_n + 1 > \nu(y) + c$ and *unknown* otherwise, while our hypothesis is that $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \phi$ is *true*.

$\boldsymbol{\phi := x.\phi_1}$ Since $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash x.\phi_1$ is *true*, $\langle \alpha^n_k(\rho), i, \nu^\alpha[x \mapsto \alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\tau_i)] \rangle \vDash \phi_1$ is *true*. By the induction hypothesis, $\forall \rho, \nu' : \rho \in \mathcal{P} \wedge \nu' \in \mathfrak{V}' : \langle \rho, i, \nu' \rangle \vDash \phi_1$ is *true*, where $\mathfrak{V}' = \{\nu' : V \nrightarrow \mathbb{Z} | \forall x : x \in V : \nu'(x) \in \gamma(\nu^\alpha[x \mapsto \alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\tau_i)]) \wedge \alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\nu'(x)) = \nu^\alpha[x \mapsto \alpha^{\bar{\tau}_n}_{\bar{\tau}_k}(\tau_i)]\}$. Moreover, for all $\nu' \in \mathfrak{V}'$, there exists $\nu \in \mathfrak{V}$ so that $\nu' = \nu[x \mapsto \tau_i]$ and vice versa. Hence $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash x.\phi_1)$ is *true*.

$\boldsymbol{\phi := \neg\phi_1}$ Since $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \neg\phi_1$ is *true*, we have that $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \phi_1$ is *false*. By the induction hypothesis $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi_1)$ is *false*. Hence $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \neg\phi_1)$ is *true*.

$\boldsymbol{\phi := \phi_1 \wedge \phi_2}$ Since $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \phi_1 \wedge \phi_2$ holds, $\min\{\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \phi_1, \langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \phi_2\}$ is *true*. Hence $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \phi_1$ is *true* and $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \phi_2$ is *true*. By the induction hypothesis $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi_1)$ is *true* and $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi_2)$ is *true*, thus $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi_1 \wedge \phi_2)$.

$\boldsymbol{\phi := \phi_1 \mathcal{U} \phi_2}$ $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \phi_1 \mathcal{U} \phi_2$ holds if $\min\{\mathring{\exists} j : j \leq i : \langle \alpha^n_k(\rho), j, \nu^\alpha \rangle \vDash \phi_2, \mathring{\forall} k' : n \leq k' < j : \langle \alpha^n_k(\rho), k', \nu^\alpha \rangle \vDash \phi_1\}$ is *true*.
Let $(\sigma_j, \tau_j) = (\sigma^*, \bar{\tau}_n{}^{\uparrow})$. Since $\langle \alpha^k_n(\rho), j, \nu^\alpha \rangle \vDash \phi_2$ is *true* then for all $\rho \in \mathcal{P}$ and all $\nu \in \mathfrak{V}$, $\langle \rho, j, \nu \rangle \vDash \phi_2$ is *true*. This happens only when $\phi_2 \equiv$ *true* otherwise $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \phi_1 \mathcal{U} \phi_2$ is not *true*. Therefore we only look at $j \leq n$.
Hence, $\langle \alpha^k_n(\rho), j, \nu^\alpha \rangle \vDash \phi_2$ is *true* for some $n \geq j \geq i$ and $\langle \alpha^k_n(\rho), k', \nu^\alpha \rangle \vDash \phi_1$ is *true* for all k' satisfying $i \leq k' < j$.
By the induction hypothesis, we have that $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, j, \nu \rangle \vDash \phi_2)$ is *true* for some $n \geq j \geq i$ and for all $i \leq k' < j$, $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, k', \nu \rangle \vDash \phi_1)$ holds. Hence, $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi_1 \mathcal{U} \phi_2)$ is *true*.

$\boldsymbol{\phi := \phi_1 \mathcal{S} \phi_2}$ Since $\langle \alpha^n_k(\rho), i, \nu^\alpha \rangle \vDash \phi_1 \mathcal{S} \phi_2$, $\mathring{\exists} j : j \leq n : \langle \alpha^n_k(\rho), j, \nu^\alpha \rangle \vDash \phi_2 \wedge \mathring{\forall} k : j < k \leq i : \langle \alpha^n_k(\rho), k', \nu^\alpha \rangle \vDash \phi_1$.
We only look at the case when $k \leq j$, since for $k > j$ and $(\sigma_j, \tau_j) = (\sigma^*, \bar{\tau}_k{}^{\downarrow})$, $\langle \langle \alpha^k_n(\rho), j, \nu^\alpha \rangle \rangle \vDash \phi_2$ is *true* iff $\phi_2 \equiv$ *true*.
Hence, $\max\{\langle \alpha^n_k(\rho), j, \nu^\alpha \rangle \vDash \phi_2 | k \leq j \leq i\}$ is *true* and $\min\{\langle \alpha^n_k(\rho), k', \nu^\alpha \rangle \vDash \phi_1 | j < k' \leq i\}$ is *true*. By the induction hypothesis, $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, j, \nu \rangle \vDash \phi_2)$ is *true* for some $j \leq i$ and $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, k', \nu \rangle \vDash \phi_1)$ is *true* for all $j < k' \leq i$. Hence $(\forall \rho, \nu : \rho \in \mathcal{P} \wedge \nu \in \mathfrak{V} : \langle \rho, i, \nu \rangle \vDash \phi_1 \mathcal{S} \phi_2)$ is *true*. □

For formulas empty valuations of clock variables, the following result follows:

Corollary 1. *Let* $\bar{\rho} = (\bar{\sigma_k}, \bar{\tau_k}) \dots (\bar{\sigma_n}, \bar{\tau_n})$ *be a finite timed word, and* ρ *an extension of* $\bar{\rho}$*. Then, for any LogLogics-formula* ϕ*, the truth value of* $\bar{\rho} \vDash \phi$ *is equal to the truth value of* $\alpha_k^n(\rho) \vDash \phi$*.*

Proof. By Theorem 6.1 the truth value of $\rho \vDash \phi$, i.e. $\langle \rho, n, \epsilon \rangle \vDash \phi$ is the same as the truth value of $\langle \alpha_k^n(\rho), n, \epsilon \rangle \vDash \phi$, thus of $\alpha_k^n(\rho) \vDash \phi$. □

Recall that Definition 6.5 determines the truth value of a *LogLogics*-formula w.r.t. a finite timed word depending on its truth values w.r.t. all possible extensions. Hence, it cannot be used to compute the truth value directly as it would involve an infinite computation. The theorem above resolves this problem by reducing the check of truth values on infinitely many extensions of the given timed word to checking the truth value on a finite object, namely the corresponding abstract timed word.

Example 3. Let us evaluate the formula $\boxminus x.(x \geq 0 \Rightarrow (p \Rightarrow \diamondsuit y.(q \wedge y \leq x + 4)))$ on the finite timed word $\bar{\rho} = ((\sigma_0, 0)(\sigma_1, 1)(\sigma_2, 1)(\sigma_3, 2)(\sigma_4, 5)(\sigma_5, 8))$ such that $\sigma_i \vdash p$ is *true* for $i = 1$ and $i = 4$, and *false* for $i \in \{0, 2, 3, 5\}$; $\sigma_i \vdash q$ is *true* for $i = 3$ and *false* for $i \in \{0, 1, 2, 4, 5\}$ (see Fig. 30). Intuitively, this formula says that whenever p was encountered during the observation period, q was encountered not later than four time units after that.

By Corollary 1 we reduce our problem to evaluation of the formula w.r.t. the abstract timed word

$$\rho^\alpha = ((\sigma^*, 0^\downarrow)(\sigma_0, 0)(\sigma_1, 1)(\sigma_2, 1)(\sigma_3, 2)(\sigma_4, 5)(\sigma_5, 8)(\sigma^*, 8^\uparrow)),$$

which is the abstraction of any extension of $\bar{\rho}$.

First, we observe that we need to minimize $\langle \rho^\alpha, i, \varepsilon \rangle \vDash x.(x \geq 0 \Rightarrow (p \Rightarrow \diamondsuit y.(q \wedge y \leq x + 4)))$ for all $i \leq 5$. This is equivalent to minimizing the value of $\langle \rho^\alpha, i, \epsilon[x \mapsto \tau_i] \rangle \vDash x \geq 0 \Rightarrow (p \Rightarrow \diamondsuit y.(q \wedge y \leq x + 4))$ for $i \leq 5$. For $i = -1$, $\tau_{-1} \geq 0$ is *false* and hence the implication is *true*. For $i \in \{0, 2, 3, 5\}$, $\langle \rho^\alpha, i, \epsilon[x \mapsto \tau_i] \rangle \vDash p$ is *false* and therefore the inner implication is *true*, and so is the outer one. The cases left are:

- $i = 4$. Since $\sigma_4 \vdash p$ is *true*, the truth value of the implication coincides with the truth value of $\langle \rho^\alpha, 4, \epsilon[x \mapsto 5] \rangle \vDash \diamondsuit y.(q \wedge y \leq x + 4)$. To determine the latter value we need to maximize the value of $\langle \rho^\alpha, j, \epsilon[x \mapsto 5] \rangle \vDash y.(q \wedge y \leq x + 4)$ for $j \geq 4$, i.e., the value of $\langle \rho^\alpha, j, \epsilon[x \mapsto 5, y \mapsto \tau_j] \rangle \vDash (q \wedge y \leq x + 4)$. For each $j \geq 4$ the value of the conjunction is the least value of $\langle \rho^\alpha, j, \epsilon[x \mapsto 5, y \mapsto \tau_j] \rangle \vDash q$ and $\langle \rho^\alpha, j, \epsilon[x \mapsto 5, y \mapsto \tau_j] \rangle \vDash y \leq x + 4$.
  If $j = 4$ or $j = 5$, $\sigma_j \vdash q$ is *false* and, hence, the value of the conjunction is *false* as well. If $j = 6$, $\sigma_j \vdash q$ is *unknown*, $\tau_6 = 8^\uparrow$ and $\langle \rho^\alpha, j, \epsilon[x \mapsto 5, y \mapsto 8^\uparrow] \rangle \vDash y \leq x + 4$ reduces to $8^\uparrow \leq_\alpha 9$ that evaluates to *unknown*. Hence, the value of the conjunction in this case is *unknown*. To determine the value of the implication for $i = 4$ we should take the maximal value, which is *unknown*, obtained for $j = 6$.

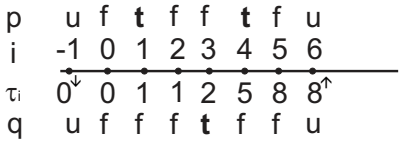


Fig. 30: A finite timed word evaluating $\boxminus x.(x \geq 0 \Rightarrow (p \Rightarrow \Diamond y.(q \wedge y \leq x + 4)))$ to *unknown*.

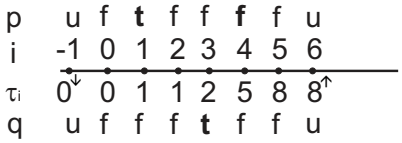


Fig. 31: A finite timed word evaluating $\boxminus x.(x \geq 0 \Rightarrow (p \Rightarrow \Diamond y.(q \wedge y \leq x + 4)))$ to *true*.

– $i = 1$. As above, since $\sigma_1 \vdash p$ is *true*, the truth value of the implication coincides with the truth value of the maximum (on $j \geq 1$) of the least of the two following values: $\langle \rho^\alpha, j, \epsilon[x \mapsto 1, y \mapsto \tau_j] \rangle \vDash q$ and $\langle \rho^\alpha, j, \epsilon[x \mapsto 1, y \mapsto \tau_j] \rangle \vDash y \leq x + 4$.

  If $j \in \{1, 2, 4, 5\}$, then $\sigma_j \vdash q$ is *false*, and so is the value of the conjunction. Since $\tau_6 = 8^\uparrow$, $\langle \rho^\alpha, j, \epsilon[x \mapsto 1, y \mapsto 8^\uparrow] \rangle \vDash y \leq x + 4$ reduces to $8^\uparrow \leq_\alpha 5$ that evaluates to *false* and the same is true for the conjunction. Finally, for $j = 3$, $\sigma_j \vdash q$ is *true* and $\tau_j = 2 \leq_\alpha 1 + 4$. Hence, both conjuncts evaluate to *true* and the conjunction as well. Hence, the maximal value is *true*, $\langle \rho^\alpha, 1, \epsilon[x \mapsto 1] \rangle \vDash \Diamond y.(q \wedge y \leq x + 4)$ evaluates to *true* and the truth value of the implication is *true*.

To find the truth value of the original formula, we need to take the least value obtained. This value is '*unknown*'.

Example 3 also explains the true meaning of *unknown*. The formula is evaluated to *unknown* due to the behavior on the boundaries of the observation sequence. Consider the finite timed word in Fig. 31 which differs from the one in Fig. 30 for $i = 4$ only. With respect to this finite timed word the response property from Example 3 is evaluated to *true*. However, if one considered $\boxminus x.(p \Rightarrow \Diamond y.(q \wedge y \leq x + 4))$ the truth value still would have been *unknown*. In such a case one might like to exclude the unknown prehistory and/or unknown future from the consideration. In fact, our formula in Example 3 excluded the prehistory. Since similar restrictions turn out to be useful for expressing inter-

esting business properties, we introduce the following short-hand notation:

$$\Box_a^b x.\phi(x) \stackrel{\text{def}}{=} \Box x.(x < a \vee x > b \vee \phi(x))$$
$$\Diamond_a^b x.\phi(x) \stackrel{\text{def}}{=} \Diamond x.(x \geq a \wedge x \leq b \wedge \phi(x))$$
$$\boxminus_a^b x.\phi(x) \stackrel{\text{def}}{=} \boxminus x.(x < a \vee x > b \vee \phi(x))$$
$$\diamondminus_a^b x.\phi(x) \stackrel{\text{def}}{=} \diamondminus x.(x \geq a \wedge x \leq b \wedge \phi(x))$$

Subscripts and superscripts of boxes and diamonds can be omitted when only one of boundaries is of interest. Using the short-hand notation formula in Example 3 can be written as $\boxminus_0 x.(p \Rightarrow \Diamond y.(q \wedge y \leq x + 4))$. Note that the limit values a and b in the short-hand notation can depend on the values of the clock variables in whose scope the corresponding temporal operator appears. This means that the formula above can be further rewritten as $\boxminus_0 x.(p \Rightarrow \Diamond_x^{x+4} y.q)$, which some users experience as more intuitive. We give more examples for the use of the abbreviations in Section 6.5.

Lemma 6.2. *Let* $\bar{\rho} = (\sigma_k, \tau_k) \ldots (\sigma_n, \tau_n)$ *be a finite timed word. Let* a restricted *LogLogics*-formula ψ *be inductively defined as:*

$$\psi := p \mid x \sim y + c \mid x \sim c \mid x.\psi \mid \mathit{false} \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \Box_a^b \psi \mid \boxminus_a^b \psi \mid \Diamond_a^b \psi \mid \diamondminus_a^b \psi$$

where $x, y \in V$, $p \in P$, $\sim$ *is one of* $<, >, \leq, \geq, =, \neq$, $c \in \mathbb{N}$ *and* $\tau_k \leq a \leq b \leq \tau_n$. *Then* $\bar{\rho} \vDash \psi$ *is evaluated to true or false.*

Note that by using restricted *LogLogics*-formulas only we obtain the logic that coincides with the logics from [1, 40].

6.4 Algorithm

In this section we present an algorithm that evaluates a given *LogLogics*-formula ϕ w.r.t. a given abstract timed word and context $\langle \rho^\alpha, i, \nu^\alpha \rangle$.

We assume the existence of two auxiliary procedures **EvalClock**(cvc, ν^α) and **EvalAtomic**$(p, a, b, \rho^\alpha, i)$, where p is an atomic proposition, cvc a clock variable comparison ($x \sim c$ or $x \sim y + c$) and a and b correspond to pre-history and future indexes. In Algorithm 6 we define the procedure **Eval** with the following parameters: a *LogLogics* formula ϕ, an abstract timed word ρ^α, a pre-history event index a, a future event index b, a current event index i, a clock valuation ν^α, a current minimum truth value *min* and a current maximum truth value *max*. The evaluation of a closed formula ϕ w.r.t. an abstract timed word $\rho^\alpha = (\sigma^*, (\tau_{a+1})^\downarrow)(\sigma_{a+1}, \tau_{a+1}) \ldots (\sigma_{b-1}, \tau_{b-1})(\sigma^*, (\tau_{b-1})^\uparrow)$ is performed by calling **Eval**$(\phi, \rho^\alpha, a, b, \mathit{last}, \varnothing, \mathit{false}, \mathit{true})$, where *last* gives the index of the last 'non-abstract' entry in ρ^α, i.e., $\mathit{last} = b - 1$. In the remainder, we assume that all the additional operators such as $\vee$ or $\boxminus$ in the formula ϕ have been replaced by their definitions.

Algorithm 6: Procedure **Eval**$(\phi, \rho^\alpha, a, b, i, \nu^\alpha, \min, \max)$

Input: $\phi, \rho^\alpha, a, b, i, \nu^\alpha, min, max$
Output: $x \in \{true, false, unknown\}$

if $min = max$ **then return** min;
else if $\phi = p$ **then** $m :=$ **EvalAtomic**$(p, \rho^\alpha, a, b, i)$;
else if $\phi = false$ **then** $m := false$;
else if $\phi = unknown$ **then** $m := unknown$;
else if $\phi = cvc$ **then** $m :=$ **EvalClock**(cvc, ν^α);
else if $\phi = x.\psi$ **then** $m :=$ **Eval**$(\psi, \rho^\alpha, a, b, i, \nu^\alpha[x \mapsto \tau_i], min, max)$;
else if $\phi = \neg\psi$ **then** $m := \neg$**Eval**$(\psi, \rho^\alpha, a, b, i, \nu^\alpha, \neg max, \neg min)$;
else if $\phi = \phi_1 \wedge \phi_2$ **then**
  $m :=$ **Eval**$(\phi_1, \rho^\alpha, a, b, i, \nu^\alpha, min, max)$;
  if $m \leq min$ **then** $m := min$ **else** $m :=$ **Eval**$(\phi_2, \rho^\alpha, a, b, i, \nu^\alpha, min, m)$
end
else if $\phi = \phi_1 \mathcal{U} \phi_2$ **then**
  $m :=$ **Eval**$(\phi_2, \rho^\alpha, a, b, i, \nu^\alpha, min, max)$;
  if $m \geq max$ **then** $m := max$;
  else
    $n :=$ **Eval**$(\phi_1, \rho^\alpha, a, b, i, \nu^\alpha, m, max)$;
    if $n > m$ *and* $i < b$ **then** $m :=$ **Eval**$(\phi, \rho^\alpha, a, b, i+1, \nu^\alpha, m, n)$;
  end
end
else if $\phi = \phi_1 \mathcal{S} \phi_2$ **then**
  $m :=$ **Eval**$(\phi_2, \rho^\alpha, a, b, i, \nu^\alpha, min, max)$;
  if $m \geq max$ **then** $m := max$;
  else
    $n :=$ **Eval**$(\phi_1, \rho^\alpha, a, b, i, \nu^\alpha, m, max)$;
    if $n > m$ *and* $i > a$ **then** $m :=$ **Eval**$(\phi, \rho^\alpha, a, b, i-1, \nu^\alpha, m, n)$
  end
end
if $m \leq min$ **then return** min;
else if $m \geq \max$ **then return** $\max$;
else return m

Depending on the form of ϕ, the procedure **Eval** recursively calls itself until the subnodes have been exhausted or $max = min$. The formula is thus evaluated as a tree with atomic propositions p and clock variable comparisons $x \sim c, x \sim y + c$ as leaves and operator symbols as other nodes. The algorithm makes a nondeterministic choice when evaluating conjunctions. A speedup may be possible by making better choices, choosing subnodes that can be evaluated fast and are likely to become *false*.

Recall that $\phi_1 \mathcal{U} \phi_2$ is *true* w.r.t. $\langle \rho^\alpha, i, \nu^\alpha \rangle$ if either ϕ_2 is *true* w.r.t. $\langle \rho^\alpha, i, \nu^\alpha \rangle$ or both ϕ_1 is *true* w.r.t. $\langle \rho^\alpha, i, \nu^\alpha \rangle$ and $\phi_1 \mathcal{U} \phi_2$ is *true* w.r.t. $\langle \rho^\alpha, i+1, \nu^\alpha \rangle$. In our three-valued case, $\langle \rho^\alpha, i, \nu^\alpha \rangle \models \phi_1 \mathcal{U} \phi_2$ is $(\langle \rho^\alpha, i, \nu^\alpha \rangle \models \phi_2 \vee (\langle \rho^\alpha, i, \nu^\alpha \rangle \models \phi_1 \wedge \langle \rho^\alpha, i+1, \nu^\alpha \rangle \models \phi_1 \mathcal{U} \phi_2))$. The value of i is limited by the length of the

word. This observation is used in the algorithm to evaluate *LogLogics*-formulas of the form $\phi_1 \mathcal{U} \phi_2$. The case of $\mathcal{S}$ is analogous.

Termination of the algorithm stems from the following fact: at each step of the computation, evaluating $\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash \phi$ is reduced to evaluating a finite number of $\langle \rho^\alpha, i_1, \nu_1 \rangle \vDash \phi_1, \ldots, \langle \rho^\alpha, i_n, \nu_n \rangle \vDash \phi_n$. The parameter n is bounded by maximum of $b - a + 1$ (cases of $\mathcal{U}$ and $\mathcal{S}$) and 2 (conjunction). Each one of the $\langle \rho^\alpha, i_j, \nu_j \rangle \vDash \phi_j$ is strictly smaller than $\langle \rho^\alpha, i, \nu^\alpha \rangle \vDash \phi$ w.r.t. the following order relation:

$$(\langle \rho^\alpha, i_1, \nu_1 \rangle \vDash \phi_1) > (\langle \rho^\alpha, i_2, \nu_2 \rangle \vDash \phi_2) \quad \text{if} \quad \begin{cases} \phi_2 \text{ is a subterm of } \phi_1, \\ \text{or } \phi_2 \text{ coincides with } \phi_1 \text{ and} \\ \quad \phi_2 = \psi_1 \mathcal{U} \psi_2 \text{ and } i_1 < i_2 \\ \quad \text{or} \quad \phi_2 = \psi_1 \mathcal{S} \psi_2 \text{ and } i_1 > i_2. \end{cases}$$

Finally, observe that the parameters *min* and *max* express the information gained so far on the range of relevant values of the subformula. By relevant values we understand those values that can influence the truth value of the supervalue. Moreover, one can show that $min \leq \mathbf{Eval}(\phi, \rho^\alpha, a, b, i, \nu^\alpha, min, max) \leq max$ for any values of the parameters.

6.5 Typical Guards of Interest

Dwyer *et al.* [49] have identified a number of property specification patterns for software verification and formalized them in LTL and CTL. Similar patterns were given in [10] for modeling processes using constraints templates which are LTL formulas and a graphical language is provided for them called DecSerFlow. In this section we analogously consider *LogLogics* guard specification patterns for business processes.

Occurrence patterns

The first group of patterns concerns the occurrence of a certain desired event, or dually, the absence of a certain undesirable event. In the most general form it requires that in a given scope a given event occurs at least a and at most b times. In particular, if $b = 0$, the event does not occur at all, and if a equals the number of time points in a scope, the event occurs throughout the entire scope. Patterns belonging to this group are occurrence, bounded occurrence, absence and universality.

Occurrence patterns allow us to check whether some event happened in a certain time interval, e.g. whether there was a transaction for a sum exceeding 5.000.000 in 2005, which we can encode as $\Diamond x.(p \wedge x \geq \text{`Jan. 1, 2005'} \wedge x \leq \text{`Dec. 31, 2005'})$, where p stands for a transaction exceeding 5.000.000. Using the notation of Section 6.3, it can be also written as $\Diamond_{\text{`Jan. 1, 2005'}}^{\text{`Dec. 31, 2005'}} x.p$. In

general, the occurrence pattern has the following form: $\Diamond_a^b x.\phi$, where a and/or b can be omitted.

Bounded occurrence is similar to the occurrence pattern but requires a certain event to occur at least k times within a scope:

$$\begin{aligned}&\Diamond x_1.(\phi(x_1) \wedge x_1 \geq a \wedge x_1 \leq b \wedge \\ &\quad \Diamond x_2.(\phi(x_2) \wedge x_2 \geq a \wedge x_2 \leq b \wedge x_2 \neq x_1 \wedge \ldots \\ &\quad\quad \Diamond x_k.(\phi(x_k) \wedge x_k \geq a \wedge x_k \leq b \wedge x_k \neq x_1 \wedge \ldots \wedge x_k \neq x_{k-1}))),\end{aligned}$$

or alternatively,

$$\begin{aligned}&\Diamond_a^b x_1.(\phi(x_1) \wedge \Diamond_a^b x_2.(\phi(x_2) \wedge x_2 \neq x_1 \wedge \ldots \\ &\Diamond_a^b x_k.(\phi(x_k) \wedge x_k \neq x_1 \wedge \ldots \wedge x_k \neq x_{k-1}))).\end{aligned}$$

Variants of this pattern require the event to occur exactly k or at most k times. Using this pattern we can e.g. express the guard checking whether there were at least three transactions for a sum exceeding 5.000.000 between January 1, 2005 and December 31, 2005.

Absence pattern is dual to the occurrence pattern and can be written as $\boxminus x.(\neg\phi(x) \vee x < a \vee x > b)$ or alternatively as $\boxminus_a^b x.\neg\phi(x)$, where ϕ denotes an event undesired between time points a and b. In this way we can check that between a and b no transaction was rejected.

Universality pattern allows to express properties that should hold throughout the period from a to b: $\boxminus x.(\phi(x) \vee x < a \vee x > b)$, i.e., $\boxminus_a^b x.\phi(x)$. A property we could express with this pattern is "between a and b all transactions were executed successfully".

Ordering patterns

The next group of patterns, called *ordering patterns*, expresses an ordering relationship between two (or more) events. Ordering patterns can be constructed from the occurrence patterns by demanding that one of them occurs in a scope within a time slot of another one.

Bounded response is an extremely common pattern, an instance of which we considered in Example 3. It allows us to express such guards as "every bill is paid within 30 days". In general, the pattern has the form $\boxminus_a^b x.(\phi_1(x) \Rightarrow \Diamond_{c(x)}^{d(x)} y.\phi_2(y))$, where $c(x)$ and $d(x)$ are timed constraints, i.e., propositional formulas over clock variable comparisons $x \sim c$ and $x \sim y + c$.

Precedence pattern requires that any occurrence of p is preceded by an occurrence of q within a scope: $\boxminus_a^b x.(\phi_1(x) \Rightarrow \Diamond_{c(x)}^{d(x)} y.\phi_2(y))$. An instance of this pattern allows us to express the guard that a loan was preceded by a credibility check with an outcome above a certain threshold.

Absence between pattern requires that between time points a and b, no r-event happens between p-event and q-event, expressed as $\boxminus_a^b x.((p \wedge \neg q \wedge \Diamond q) \rightarrow (\neg r \mathcal{U} q))$. An example of a guard would be "no credit card transactions took

place between the card issue and the report that the card was not received by the legal owner".

Compound patterns, forming the last group of patterns, can be constructed from the patterns above by means of conjunction and disjunction.

The advantage of using *LogLogics* is that it can express past and time properties. For instance the precedence pattern is more succinct than its counterpart having future temporal logic operators [10]. Time constraints can be analogously integrated in the DecSerFlow patterns.

6.6 Conclusion

In this chapter we have proposed a logic that works on finite traces and is appropriate for specifying guards in models of history-dependent processes. Since at any given moment in time information is finite and inherently incomplete, we had to adapt existing timed temporal logics, which resulted in a three-valued logic, *LogLogics*, presented above. Although the straightforward application of the definition of the *LogLogics* semantics gives rise to a procedure with an infinite number of checks, we have shown that a check of the truth of an *LogLogics*-formula can be reduced to a check of its truth value on a finite *abstraction*. Moreover, we have also shown how guard patterns common for business processes can be expressed in *LogLogics*.

7

Conclusion and Future Work

In this chapter, we summarize the main contributions to this thesis and pose some questions for future work.

This work was carried out within the project Modeling and Verification of Business Processes[7] (MoVeBP), which focused on extending the expressive power of the Petri net modeling framework for business processes in such a way that formal verification of properties with standard techniques like model checking would still be possible. For classical Petri nets, with indistinguishable ("black") tokens, there are several good verification techniques available, but the expressive power, and more importantly, the modeling convenience of this class of nets is low compared to the other end of the spectrum: colored Petri nets. Colored Petri nets offer a comfortable modeling power at the expense of limited verification possibilities: by simulation, searching a finite part of the state space and a limited number of verification of behavioral properties when the state space is infinite (abstractions and structural techniques, for instance).

In this thesis we addressed specific features of business process languages like batch processing of workflow cases, data and timing, and different flexibility aspects for underspecified processes, where parts of some process are not given at design time but provided later during runtime or corrections of (un)desired or (un)anticipated behavior at certain decision points in the running process are required.

Soundness is the central correctness property of workflows considered here, which means that all runs of the workflow can always terminate, i.e. reach a final state and the all its actions (transitions) can eventually be executed. For workflow with specific features, we have considered several versions of this correctness criterion and described means of verifying such properties.

Generalized soundness

We considered the parametrized notion of soundness called generalized soundness which corresponds to the processing of batches of cases by a workflow and proper termination of all started cases. A workflow net with an arbitrary initial marking can be considered as an abstraction of the same workflow net having multiple case identifiers for the initial marking.

We described an improved procedure for deciding generalized soundness property for workflow nets based on abstraction. We show how this procedure can be used to prove soundness by reduction of generalized sound subworkflow

[7] financially supported by NWO Open competition, project number 621.000.315.

nets. Since the generalized soundness check is expensive, we show how we can use transformation techniques which preserve liveness and boundedness of the workflow and incorporate these transformation in the soundness check for large workflows.

Our approach has some limitations. One such limitation refers to the fact that the algorithm for finding the set of minimal markings does not rely on finding the solutions to the set of integer linear equations, but simply incrementally searches these solutions through the integer points of some polytope, and therefore is not optimal. An option to alleviate this disadvantage would be to find solutions to the set of constraints given above in an incremental way until we compute all minimal markings using integer linear programming techniques. Another limitation is still the state space explosion caused by the backward reachability checks, which can be very expensive. An option would be to incorporate some symbolic methods of storing the backward set of reachable markings.

For future work, we are interested in the verification of temporal logic properties of Petri nets (not necessarily WF-nets) with using such a reduction technique. The idea can also be applied to build sound by construction nets in a hierarchical way similarly to Vogler's refinement by modules [139, 140]. Another direction for future work is finding subclasses of nets for which counterexamples can be directly derived, similarly to the way reachable markings are derived from solving equations as in [130].

As other directions for future work, we can mention possible extensions of the algorithm for finding minimal markings with capabilities for finding counterexamples. These results can be further used to find similar properties of extensions of workflow nets, for instance if nets have multiple initial and final places and the initial and final markings satisfy certain parametrized constraints.

Modeling and verification of adaptive workflow

To model adaptive workflows, we considered Adaptive Workflow Nets, a subclass of nested nets that allows more comfort and expressive power for modeling adaptation and exception handling in workflow nets. Flexibility was achieved by allowing transitions that create new nets out of the existing ones. Therefore, nets with completely new structure could be created at run time.

We defined two properties for adaptive nets: soundness which refers to their correctness, and circumspectness, referring to their robustness (exception handling). Soundness is defined as the property by which a proper final marking (state) can be reached from any marking which is reachable by firing non-exceptional transitions from the initial marking. Circumspectness is defined as the property by which the upper layer is always ready to handle any exception that is triggered in a lower layer.

Using abstraction, we reduced the problem of verifying these properties from a verification problem for a net with an infinite state space to a check

of properties of a net with finite state space. In particular, we extracted the essential information from adaptive nets, which is captured by the transitions used to model the exceptions. We described several compositional procedures for the verification of adaptive workflow nets.

Our formalism is limited by the hierarchical "separation of concerns" mechnism. We only give necessary and sufficient conditions for the verification of soundness and properties related to proper exception handling such as circumspectness for some subclasses of adaptive nets. It would be interesting to find similar characterizations for extensions of this formalism taking into account resources and timing constraints. A step towards such characterizations was done in [113], where resource nets are net tokens that are synchronized horizontally with the main workflow. It would be interesting to map resource patterns [122] to resource token nets and study the correctness of the resulting nets.

Tools for modeling and simulation of the "nets in nets" formalism are RENEW [82] and MAUDE [70]. Renew offers no verification capabilities while MAUDE only allows for verification in case the process has a finite state space. However, as we have shown sound models can still have infinite state space. For future work, it would be interesting to consider building a prototype for editing, simulating and verification adaptive nets as a plug-in of Yasper [63]. Furthermore, the exception handling technique presented here can be used as a formal alternative to nested interruptible regions in UML activity diagrams [54] and the BPEL exception handling mechanism [71].

Modeling and verification of time and data-dependent business processes

In Chapter 5, we considered ARIS Toolset [73]. Aris is used for modeling and analyzing business processes and is frequently used in industry. The modeling language of this framework is extended Event-driven Process Chains (eEPC). eEPCs are intensively used in practice, although their semantics is not fully understood by designers. eEPCs do not only describe the control flow of processes by means of partial description of states (events), activities (functions) and logical connector which coordinate the control flow, but also the data, material resources, personnel resources and timing information needed for achieving the business goal. Time is an important construct in eEPCs appearing as timeouts and function durations, which are particularly important in resource management.

We provided a formal definition of the eEPC semantics in terms of a transition system. Since eEPCs and colored Petri nets have a similar structure and graphical layout and resources and data in eEPCs have limited domains a natural choice for the verification of eEPCs is their translation is colored Petri nets. Timed Colored Petri Nets (TCPNs) are supported by a dedicated simulation and verification environment (CPN Tools) and the errors in the eEPC can be easily identified in the TCPN translation.

In order to verify eEPCs, we have defined a translation of this formalism to TCPNs by means of patterns that correspond to the possible rules occurring in an eEPC. The semantics can be further used as a basis for behavioral (functional) verification of eEPCs using different (more efficient) model checkers which have more extensive verification features, e.g. MARIA [97], SPIN.

Modeling history-dependent business processes

History dependent processes have emerged as an important feature of case-handling systems in enterprise information systems, which have the recording of logs as a common feature. Event traces are archived not only for analyzing performance of executed processes, but also for decision making during the execution of processes and for creation of new processes based on the information from the logs.

We proposed a logic that works on finite traces (logs) and is appropriate for specifying guards in models of history-dependent processes and for analyzing logs. Since at any given moment in time the log is finite and the process is normally still running, we had to adapt existing timed temporal logics, which resulted in a three-valued logics: *LogLogics*. Although the straightforward application of the definition of the *LogLogics* semantics to the check of the truth value of formulas involved an infinite number of checks, we showed that the satisfiability of a *LogLogics*-formula can be reduced to the satisfiability on a finite *abstraction*. We showed how to express guard patterns for business processes in *LogLogics*.

For future work, it would be interesting to create a simple textual language for working with patterns targeted at non-specialists and to build a tool for checking *LogLogics*-formulas on history logs. The ultimate goal is to integrate the logic into existing workflow modeling frameworks, in particular in adaptive workflows [62].

Another line of research can be the investigation of different behavioral properties of Petri nets having *LogLogics*-formulas as guards. Our logic can be particularly useful for processes specified in a compositional manner, where the behavior of the environment components is partially known and where corrections of the behavior of such components is needed to ensure nets are functioning in a reliable manner as a whole.

Bibliography

[1] W. M. P. van der Aalst, H. T. de Beer, and B. F. van Dongen. Process mining and verification of properties: An approach based on temporal logic. In *OTM Conferences (1)*, volume 3760 of *LNCS*, pages 130–147. Springer, 2005. (Cited on pages 118 and 134.)

[2] W. M. P. van der Aalst. Formalization and verification of event-driven process chains. *Information and Software Technology*, 41(10):639–650, 1999. (Cited on pages 6, 8, 78 and 115.)

[3] W. M. P. van der Aalst. A class of Petri nets for modeling and analyzing business processes. Technical Report 26, Eindhoven University of Technolog, 1995. (Cited on pages 4 and 28.)

[4] W. M. P. van der Aalst. Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change. *Information Systems Frontiers*, 3(3):297–317, 2001. (Cited on page 7.)

[5] W. M. P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theor. Comput. Sci.*, 270(1-2):125–203, 2002. (Cited on pages 8 and 42.)

[6] W. M. P. van der Aalst, T. Basten, H. M. W. Verbeek, P. A. C. Verkoulen, and M. Voorhoeve. Adaptive workflow-on the interplay between flexibility and support. In *ICEIS*, pages 353–360, 1999. (Cited on page 42.)

[7] W. M. P. van der Aalst, J. Desel, and E. Kindler. On the semantics of EPCs: A vicious circle. In *EPK'02*, pages 71–79, 2002. (Cited on page 115.)

[8] W. M. P. van der Aalst, B. F. van Dongen, C. W. Günther, R. S. Mans, A. K. A. de Medeiros, A. Rozinat, V. Rubin, M. Song, H. M. W. Verbeek, and A. J. M. M. Weijters. ProM 4.0: Comprehensive Support for *real* Process Analysis. In *ICATPN*, volume 4546 of *LNCS*, pages 484–494. Springer, 2007. (Cited on page 6.)

[9] W. M. P. van der Aalst. Verification of workflow nets. In *Proc. of ICATPN'97*, volume 1248 of *LNCS*, pages 407–426. Springer, 1997. (Cited on pages 4, 21, 48 and 49.)

[10] W. M. P. van der Aalst and M. Pesic. DecSerFlow: Towards a truly declarative service flow language. In *The Role of Business Processes in Service Oriented Architectures*, number 06291 in Dagstuhl Seminar Proceedings. IBFI, 2006. (Cited on pages 74, 136 and 138.)

[11] W. M. P. van der Aalst and K. M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002. (Cited on pages 3, 21 and 118.)

[12] W. M. P. van der Aalst, D. Moldt, R. Valk, and F. Wienberg. Enacting Interorganizational Workflows Using Nets in Nets. In *Proc. of Workflow Management'99*, pages 117–136, 1999. (Cited on page 74.)

[13] W. M. P. van der Aalst, P. J. N. de Crom, R. H. M. J. Goverde, K. M. van Hee, W. J. Hofman, H. A. Reijers, and R. A. van der Toorn. ExSpect 6.4: An executable specification tool for hierarchical colored Petri nets. In *ICATPN'00*, volume 1825 of *LNCS*, pages 455–464. Springer, 2000. (Cited on page 8.)

[14] M. Adams, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Facilitating flexibility and dynamic exception handling in workflows through worklets. In *CAiSE'05, CAiSE Short Paper Proceedings*, volume 161, pages 45–50, 2005. (Cited on page 42.)

[15] R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *REX Workshop*, volume 600 of *LNCS*, pages 74–106. Springer, 1991. (Cited on pages 118 and 119.)

[16] R. Alur and T. A. Henzinger. A really temporal logic. *J. ACM*, 41(1): 181–204, 1994. (Cited on pages 118, 119 and 120.)

[17] A. Alves, A. Arkin, S. Askary, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, C. K. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu. Web services business process execution language version 2.0, 2006. (Cited on page 2.)

[18] R. Bagnara, P. Hill, and E. Zaffanela. The Parma Polyhedra Library users manual. `www.cs.unipr.it/ppl/Documentation`. (Cited on page 38.)

[19] R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Quaderno 457, Dipartimento di Matematica, Università di Parma, Italy, 2006. (Cited on page 144.)

[20] R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 2008. To appear. Journal version of [19]. (Cited on page 38.)

[21] G. Berthelot. Checking properties of nets using transformation. In *ATPN'85*, volume 222 of *LNCS*, pages 19–40. Springer, 1986. (Cited on page 4.)

[22] G. Berthelot. *Verification de Reseaux de Petri*. PhD thesis, Universite Pierre et Marie Curie (Paris), 1978. (Cited on pages 4 and 37.)

[23] E. Best and M. Koutny. Process algebra: A Petri-net-oriented tutorial. In *LNCS*, volume 3098, pages 180–209, 2003. (Cited on page 51.)

[24] O. Biberstein, D. Buchs, and N. Guelfi. Object-oriented nets with algebraic specifications: The CO-OPN/2 formalism. In *Concurrent Object-Oriented Programming and Petri Nets*, volume 2001 of *LNCS*, pages 73–130. Springer, 2001. (Cited on pages 3 and 74.)

[25] P. Bouyer, F. Chevalier, and N. Markey. On the expressiveness of TPTL and MTL. In *FSTTCS'05*, volume 3821 of *LNCS*. Springer, 2005. (Cited on pages 119 and 120.)

[26] W. Brauer, R. Gold, and W. Vogler. A survey of behaviour and equivalence preserving refinements of Petri nets. In *APN'90*, pages 1–46. Springer-Verlag New York, Inc., 1991. (Cited on page 7.)

[27] G. Chehaibar. Replacement of open interface subnets and stable state transformation equivalence. In *ICATPN'93*, pages 1–25. Springer-Verlag, 1993. (Cited on page 7.)

[28] G. Chehaibar. Use of reentrant nets in modular analysis of colored nets. *LNCS; Advances in Petri Nets*, 524:58–77, 1991. (Cited on page 7.)

[29] A. Cheng, S. Christensen, and K.H. Mortensen. Model Checking Coloured Petri Nets Exploiting Strongly Connected Components. In *WODES'96*, pages 169–177, 1996. (Cited on page 115.)

[30] S. Christensen and L. Petrucci. Modular analysis of Petri nets. *Computer Journal*, 43(3):224–242, 2000. (Cited on page 3.)

[31] S. Christensen, L.M. Kristensen, and T. Mailund. Condensed state spaces for timed Petri nets. In *ICATPN'01*, volume 2075 of *LNCS*, pages 101–120. Springer, 2001. (Cited on page 108.)

[32] R. Clarisó, E. Rodriguez-Carbonell, and J. Cortadella. Derivation of non-structural invariants of Petri nets using abstract interpretation. In *ICATPN'05*, volume 3536 of *LNCS*, pages 188–207. Springer-Verlag, 2005. (Cited on page 5.)

[33] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1982. (Cited on page 4.)

[34] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000. (Cited on page 4.)

[35] F. Commoner. *Deadlocks in Petri Nets*. Applied Data Research, Inc., Wakefield, Massachusetts, Report CA-7206-2311, 1972. (Cited on page 19.)

[36] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL'77*, pages 238–252. ACM Press, 1977. (Cited on pages 4 and 123.)

[37] CPN Tools. `http://www.daimi.au.dk/CPNtools`. (Cited on pages 8, 78, 92 and 108.)

[38] N. Cuntz and E. Kindler. On the semantics of EPCs: Efficient calculation and simulation. In *EPK'04*, GI, pages 7–26, Bonn, 2004. (Cited on page 115.)

[39] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(2):253–291, 1997. (Cited on pages 4 and 123.)

[40] Z. Dang, T. Bultan, O. H. Ibarra, and R. A. Kemmerer. Past pushdown timed automata and safety verification. *Theor. Comput. Sci.*, 313(1): 57–71, 2004. (Cited on pages 118 and 134.)

[41] R. Davis. *Business Process Modeling with ARIS: A Practical Guide*. Springer-Verlag, 2001. (Cited on pages 2, 8 and 79.)

[42] D. de Frutos Escrig and C. Johnen. Decidability of home space property. Technical report, Univ. de Paris-Sud, Centre d'Orsay, Laboratoire de Recherche en Informatique Report LRI–503, July 1989. (Cited on page 29.)

[43] J. Dehnert. *A Methodology for Workflow Modeling - From business process modeling towards sound workflow specification*. PhD thesis, TU Berlin, 2003. (Cited on pages 8, 49, 78 and 115.)

[44] J. Desel and J. Esparza. *Free Choice Petri nets.*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995. (Cited on pages 4, 19 and 20.)

[45] J. Desel, W. Reisig, and G. Rozenberg, editors. *Advances in Petri Nets*, volume 3098 of *LNCS*, 2004. Springer. (Cited on page 17.)

[46] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distict prime factors. *American Journal of Mathematics*, 35(4):413–422, 1913. (Cited on page 14.)

[47] B. van Dongen. *Process mining and verification*. Dissertation, Technical University Eindhoven, 2007. (Cited on pages 108 and 115.)

[48] M. Dumas and A. H. M. ter Hofstede. UML activity diagrams as a workflow specification language. In *UML*, volume 2185 of *LNCS*, pages 76–90. Springer, 2001. (Cited on pages 2 and 8.)

[49] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE'99*, pages 411–420. IEEE Computer Society Press, 1999. (Cited on page 136.)

[50] C. A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *COOCS'95*, pages 10–21. ACM, 1995. (Cited on page 7.)

[51] J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Journal of Informatik Processing and Cybernetics*, 30(3):143–160, 1994. (Cited on page 4.)

[52] A. Fent, H. Reiter, and B. Freitag. Design for change: Evolving workflow specifications in ULTRAflow. In *Advanced Information Systems Engineering, CAiSE'02*, volume 2348 of *LNCS*, pages 516–534. Springer, 2002. (Cited on page 42.)

[53] F. V. Fossela, R. Komaki, and G. L. Walsh. Small-Cell Lung Cancer. `http://utm-ext01a.mdacc.tmc.edu/mda/cm/CWTGuide.nsf/LuHTML/`, 2000. (Cited on page 44.)

[54] B. Gallina, N. Guelfi, and A. Mammar. Structuring business nested processes using UML 2.0 activity diagrams and translating into XPDL. In *XML4BPN XML Integration and Transformation for Business Process Management*, pages 281–296. GITO-Verlag, 2006. (Cited on page 141.)

[55] P. Ganty, J.-F. Raskin, and L. Van Begin. A complete abstract interpretation framework for coverability properties of WSTS. In *VMCAI'06*, volume 3855 of *LNCS*, pages 49–64. Springer, 2006. (Cited on page 5.)

[56] R. J. van Glabbeek. The linear time - branching time spectrum ii. In *CONCUR'93*, volume 715 of *LNCS*, pages 66–81. Springer, 1993. (Cited on page 15.)

[57] S. Haddad and J.-F. Pradat-Peyre. New efficient Petri nets reductions for parallel programs verification. *Parallel Processing Letters*, 16(1):101–116, 2006. (Cited on page 4.)

[58] M. Han, T. Thiery, and X. Song. Managing exceptions in the medical workflow systems. In *ICSE'06*, pages 741–750. ACM Press, 2006. (Cited on pages 6 and 8.)

[59] K. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and separability of workflow nets in the stepwise refinement approach. In *Proc. of ICATPN'03*, volume 2679 of *LNCS*, pages 337–356, 2003. (Cited on pages 4, 5, 7, 28, 38 and 39.)

[60] K. van Hee, N. Sidorova, and M. Voorhoeve. Generalized soundness of workflow nets is decidable. In *Proc. of ICATPN'04*, volume 3099 of *LNCS*, pages 197–216, 2004. (Cited on pages 4, 5, 6, 7, 9, 27, 28, 29, 30, 31, 32, 39 and 48.)

[61] K. van Hee, O. Oanea, and N. Sidorova. Colored Petri nets to verify extended event-driven process chains. In *OTM Conferences (1)*, volume 3760 of *LNCS*, pages 183–201. Springer, 2005. (Cited on page 77.)

[62] K. van Hee, I. Lomazova, O. Oanea, A. Serebrenik, N. Sidorova, and M. Voorhoeve. Nested Nets for Adaptive Systems. In *ICATPN'06*, volume 4024 of *LNCS*, pages 241–260, 2006. (Cited on pages 41, 42, 43, 55 and 142.)

[63] K. van Hee, O. Oanea, R. Post, L. Somers, and J. M. van der Werf. Yasper: a tool for workflow modeling and analysis. In *ACSD'06*, pages 279–282. IEEE, 2006. (Cited on pages 8, 38 and 141.)

[64] K. van Hee, O. Oanea, N. Sidorova, and M. Voorhoeve. Verifying generalized soundness for workflow nets. In *PSI'06*, volume 4378 of *LNCS*, pages 235–247. Springer, 2007. (Cited on pages 27 and 74.)

[65] K. van Hee, H. Schonenberg, A. Serebrenik, N. Sidorova, and J. M. van der Werf. Adaptive Workflows for Healthcare Information Systems. In *Proc. of ProHealth'07*, pages 41–52, 2007. (Cited on page 74.)

[66] K. van Hee, A. Serebrenik, N. Sidorova, and W. M. P. van der Aalst. History-dependent Petri nets. In *ICATPN'07*, volume 4546 of *LNCS*, pages 164–183. Springer, 2007. (Cited on page 8.)

[67] K. van Hee, I. A. Lomazova, O. Oanea, A. Serebrenik, N. Sidorova, and M. Voorhoeve. Checking properties of adaptive workflow nets. In *CS&P'06*, pages 92–103, 2006. (Cited on page 41.)

[68] K. van Hee, I. A. Lomazova, O. Oanea, A. Serebrenik, N. Sidorova, and M. Voorhoeve. Checking properties of adaptive workflow nets. *Fund. Inform.*, 79(3-4):347–362, 2007. (Cited on pages 41 and 43.)

[69] K. van Hee, O. Oanea, A. Serebrenik, N. Sidorova, and M. Voorhoeve. Loglogics: A logic for history-dependent business processes. *Sci. Comput. Program.*, 65(1):30–40, 2007. (Cited on page 117.)

[70] A. Hicheur, K. Barkaoui, and N. Boudiaf. Modeling workflows with recursive ECATNets. In *SYNASC'06*, pages 389–398. IEEE Computer Society, 2006. (Cited on page 141.)

[71] S. Hinz, K. Schmidt, and C. Stahl. Transforming BPEL to Petri nets. In *BPM'05*, volume 3649 of *LNCS*, pages 220–235. Springer, 2005. (Cited on pages 8 and 141.)

[72] K. Hoffmann, H. Ehrig, and T. Mossakowski. High-level nets with nets and rules as tokens. In *ICATPN'05*, volume 3536 of *LNCS*, pages 268–288. Springer, 2005. (Cited on page 74.)

[73] *ARIS Methods Manual*. IDS Scheer AG, 2003. (Cited on pages 79, 116 and 141.)

[74] K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical*. Springer-Verlag, 1992. (Cited on pages 3, 22, 23, 78 and 92.)

[75] K. Jensen, L. M. Kristensen, and L. Wells. Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems. *STTT*, 9(3-4): 213–254, 2007. (Cited on pages 78 and 108.)

[76] G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation: Iterative Process Prototyping*. Addison-Wesley, 1998. (Cited on page 78.)

[77] G. Keller, K. G. Nüttgens, and A.-W Scheer. Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK). Technical report, Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, Saarbrücken, 1992. (Cited on pages 78 and 79.)

[78] E. Kindler. On the semantics of EPCs: Resolving the vicious circle. *Data Knowl. Eng.*, 56(1):23–40, 2006. (Cited on page 6.)

[79] E. Kindler. On the semantics of EPCs: A framework for resolving a vicious circle. In *BMP'04*, volume 3080 of *LNCS*, pages 82–97. Springer, 2004. (Cited on page 115.)

[80] S. C. Kleene. *Introduction to Metamathematics*. Van Nostrand, Princeton, 1952. (Cited on page 123.)

[81] M. Köhler and H. Rölke. Reference and value semantics are equivalent for ordinary object Petri nets. In *ICATPN'05*, volume 3536 of *LNCS*, pages 309–328. Springer, 2005. (Cited on page 74.)

[82] M. Köhler and H. Rölke. Dynamic transition refinement. In *FOCLASA'06*, 2006. to appear in ENCTS. (Cited on pages 74 and 141.)

[83] P. Koksal, S. N. Arpinar, and A. Dogac. Workflow history management. *SIGMOD Record*, 27(1):67–75, 1998. (Cited on page 8.)

[84] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990. (Cited on pages 118 and 119.)

[85] C. A. Lakos. From coloured Petri nets to object Petri nets. In *ICATPN*, volume 935 of *LNCS*, pages 278–297, 1995. (Cited on pages 3 and 74.)

[86] P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event-Driven Process Chains. In *ICATPN'98*, volume 1420 of *LNCS*, pages 286–305. Springer, 1998. (Cited on pages 6, 78 and 115.)

[87] T. Latvala. Model checking LTL properties of high-level Petri nets with fairness constraints. In *ICATPN'01*, volume 2075 of *LNCS*, pages 242–262. Springer, 2001. (Cited on page 4.)

[88] T. Latvala and M. Mäkelä. LTL model checking for modular Petri nets. In *ICATPN'04*, volume 3099 of *LNCS*, pages 298–311. Springer, 2004. (Cited on page 7.)

[89] K. Lautenbach. *Liveness in Petri Nets.* St. Augustin: Gesellschaft fr Mathematik und Datenverarbeitung Bonn, Interner Bericht ISF-75-02.1, 1975. (Cited on page 20.)

[90] G. Lewis and C. Lakos. Incremental state space construction for coloured Petri nets. In *ICATPN'01*, volume 2075 of *LNCS*, pages 263–282. Springer, 2001. (Cited on page 7.)

[91] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, 1997. (Cited on page 12.)

[92] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–44, 1995. (Cited on page 123.)

[93] I. A. Lomazova. Nested Petri nets - a formalism for specification and verification of multi-agent distributed systems. *Fundam. Inform.*, 43(1-4):195–214, 2000. (Cited on pages 3 and 42.)

[94] I. A. Lomazova. Nested Petri nets: Multi-level and recursive systems. *Fundam. Inform.*, 47(3-4):283–293, 2001. (Cited on pages 42, 43, 55 and 74.)

[95] I. A. Lomazova and Ph. Schnoebelen. Some decidability results for nested Petri nets. In *Ershov Memorial Conference*, volume 1755 of *LNCS*, pages 208–220. Springer, 1999. (Cited on pages 42 and 74.)

[96] Z. Luo, A. P. Sheth, K. Kochut, and J. A. Miller. Exception handling in workflow systems. *Applied Intelligence*, 13(2):125–147, 2000. (Cited on pages 6 and 8.)

[97] M. Mäkelä. Maria: Modular reachability analyser for algebraic system nets. In *ICATPN'02*, number 2360 in LNCS, pages 434–444, 2002. (Cited on page 142.)

[98] A. Martens. *Verteilte Geschätsprozesse - Modellierung und Verifikation mit Hilfe von Web Services.* Dissertation, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, 2003. (Cited on page 49.)

[99] E. W. Mayr. An algorithm for the general Petri net reachability problem. In *STOC'81*, pages 238–246. ACM, 1981. (Cited on page 29.)

[100] J. Mendling. *Detection and Prediction of Errors in EPC Business Process Models.* Dissertation, Vienna University of Economics and Business Administration, 2007. (Cited on page 115.)

[101] D. Moldt and F. Wienberg. Multi-agent-systems based on coloured Petri nets. In *ICATPN'97*, volume 1248 of LNCS, pages 82–101, Berlin, Germany, 1997. Springer-Verlag. (Cited on page 74.)

[102] O. Morikawa. Extended Gentzen-type formulations of two temporal logics based on incomplete knowledge systems. *Notre Dame Journal of Formal Logic*, 42(1):55–64, 2001. (Cited on page 118.)

[103] T. Murata. State equatation, controllability, and maximal matchings of Petri nets. *IEEE Trans. Autom. Contr.*, 22(3):412–416, June 1977. (Cited on page 19.)

[104] T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, volume 77(4), pages 541–580, April 1989. (Cited on pages 17, 37 and 38.)

[105] P. Muth, J. Weisenfels, M. Gillmann, and G. Weikum. Workflow history management in virtual enterprises using a light-weight workflow management system. In *RIDE*, pages 148–155, 1999. (Cited on page 8.)

[106] A. Nakamura. On a three-valued logic based on incomplete knowledge systems. Technical Report 1, Japan Research Group of Multiple-valued Logic, The Institute of Electronics, Information and Communication Engineers, 1995. (Cited on page 118.)

[107] M. Nüttgens and F. J. Rump. Syntax und Semantik Ereignisgesteuerter Processketten (EPK). In J. Desel and M. Weske, editors, *Promise'02*, volume LNI P-21, pages 64–77, 2002. (Cited on page 115.)

[108] M. Peleg, A. Boxwala, S. Tu, D. Wang, O. Ogunyemi, and Q. Zengh. Guideline interchange format 3.5 technical specification. InterMed Project, 2004. (Cited on pages 2 and 43.)

[109] M. Pesic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops*, volume 4103 of *LNCS*, pages 169–180. Springer, 2006. (Cited on page 8.)

[110] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, 1981. (Cited on pages 3 and 17.)

[111] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut fur Instrumentelle Mathematik, Bonn, 1962. (Cited on page 17.)

[112] L. Ping, H. Hao, and L. Jian. On 1-soundness and soundness of workflow nets. In *MOCA'04*, pages 21–36, 2004. (Cited on page 29.)

[113] O. Prisecaru. Resource workflow nets: a Petri net formalism for workflow modeling. In *MSVVEIS'07*, pages 11–20, 2007. (Cited on page 141.)

[114] S. Quaglini, M. Stefanelli, A. Cavallini, G. Micieli, C. Fassino, and C. Mossa. Guideline-based careflow systems. *Artificial Intelligence in Medicine*, 20(1):5–22, 2000. (Cited on page 43.)

[115] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In *Symposium on Programming*, volume 137 of *LNCS*, pages 337–351. Springer, 1982. (Cited on page 4.)

[116] J.-F. Raskin. *Logics, Automata and Classical Theories for Deciding Real-Time*. PhD thesis, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, 1999. (Cited on page 119.)

[117] H. A. Reijers and W. M. P. van der Aalst. Short-term simulation: Bridging the gap between operational control and strategic decision making. In *IASTED International Conference on Modelling and Simulation*, pages 417–421, 1999. (Cited on page 6.)

[118] W. Reisig. *Petri Nets.*, volume 4. Springer-Verlag EATCS Monographs on Theoretical Computer Science, 1985. (Cited on pages 3, 17 and 59.)

[119] C. Reutenauer. *The mathematics of Petri nets*. Prentice-Hall, Inc., 1990. (Cited on page 29.)

[120] S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems - a survey. *Data Knowl. Eng.*, 50(1):9–34, 2004. (Cited on pages 42 and 118.)

[121] S. Rinderle, M. Reichert, and P. Dadam. Flexible support of team processes by adaptive workflow systems. *Distributed and Parallel Databases*, 16(1):91–116, 2004. (Cited on pages 7 and 8.)

[122] N. Russell, W.M.P. van der Aalst, A. H. M. ter Hofstede, and D. Edmond. Workflow resource patterns: Identification, representation and tool support. In *CAiSE'05*, volume 3520 of *LNCS*, pages 216–232. Springer, 2005. (Cited on pages 118 and 141.)

[123] A.-W. Scheer. *ARIS : business process modeling*. Springer-Verlag, Berlin, 2nd edition, 1998. (Cited on pages 78 and 79.)

[124] K. Schmidt. LoLA: A low level analyser. In Nielsen, M. and Simpson, D., editors, *ICATPN'00*, volume 1825, pages 465–474. Springer-Verlag, 2000. (Cited on page 8.)

[125] K. Schmidt. *Explicit State Space Verification*. Habilitationsschrift, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, December 2002. (Cited on page 4.)

[126] Ph. Schnoebelen and N. Sidorova. Bisimulation and the reduction of Petri nets. In *ICATPN'00*, volume 1825 of *LNCS*, pages 409–423. Springer, 2000. (Cited on page 4.)

[127] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience series in discrete mathematics. John Wiley & Sons, 1986. (Cited on pages 24, 25 and 32.)

[128] P. Starke. *Analyse von Petri-Netz-Modellen*. Teubner, 1990. (Cited on page 20.)

[129] I. Suzuki and T. Murata. A method for stepwise refinement and abstraction of Petri nets. *J. Comput. Syst. Sci.*, 27(1):51–76, 1983. (Cited on page 7.)

[130] E. Teruel and M. Silva. Liveness and home states in equal conflict systems. In *ATPN'93*, volume 691 of *LNCS*, pages 415–432. Springer, 1993. (Cited on pages 35 and 140.)

[131] P. Thati and G. Rosu. Monitoring algorithms for metric temporal logic specifications. *Electr. Notes Theor. Comput. Sci.*, 113:145–162, 2005. (Cited on page 119.)

[132] F. L. Tiplea and D. C. Marinescu. Structural soundness for workflow nets is decidable. *Information Processing Letters*, 96(2):54–58, 2005. (Cited on page 29.)

[133] F. L. Tiplea and D.C. Marinescu. Structural soundness for workflow nets is decidable. *Information Processing Letters*, 96(2):41–80, 2005. (Cited on page 4.)

[134] F. L. Tiplea and A. Tiplea. Instantiating nets and their applications to workflow nets. In *SYNASC'05*, pages 25–29. IEEE, 2005. (Cited on pages 4 and 29.)

[135] R. Valette. Analysis of Petri nets by stepwise refinements. *J. Comput. Syst. Sci.*, 18(1):35–46, 1979. (Cited on page 7.)

[136] R. Valk. Object Petri Nets: Using the nets-within-nets paradigm. *Lectures on Concurrency and Petri Nets: Advances in Petri Nets, volume 3098 of LNCS*, pages 819–848, 2004. (Cited on pages 3, 42 and 74.)

[137] Rob J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In *IFIP Congress*, pages 613–618, 1989. (Cited on page 108.)

[138] H. M. W. Verbeek, T. Basten, and W. M. P. van der Aalst. Diagnosing workflow processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001. (Cited on pages 7, 8 and 49.)

[139] W. Vogler. Behaviour preserving refinement of Petri nets. In *WG*, volume 246 of *LNCS*, pages 82–93. Springer, 1986. (Cited on page 140.)

[140] W. Vogler. *Modular Construction and Partial Order Semantics of Petri Nets*, volume 625 of *LNCS*. Springer-Verlag, 1992. (Cited on page 140.)

[141] J. M. van der Werf. Analysis of well-formedness and soundness by reduction techniques and their implementation. Master's thesis, Technical University Eindhoven, 2006. (Cited on page 38.)

[142] M. Weske. Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In *HICSS'01*, 2001. (Cited on page 7.)

[143] S. A. White. Workflow patterns with BPMN and UML. Technical report, IBM, 2004. (Cited on page 2.)

[144] YASPER. Petri net editor. `www.yasper.org`. (Cited on page 38.)

Index

Summary

In this thesis we focus on improving current modeling and verification techniques for complex business processes. The objective of the thesis is to consider several aspects of real-life business processes and give specific solutions to cope with their complexity.

In particular, we address verification of a proper termination property for workflows, called *generalized soundness*. We give a new decision procedure for generalized soundness that improves the original decision procedure. The new decision procedure reports on the decidability status of generalized soundness and returns a counterexample in case the workflow net is not generalized sound. We report on experimental results obtained with the prototype implementation we made and describe how to verify large workflows compositionally, using reduction rules.

Next, we concentrate on modeling and verification of *adaptive workflows* — workflows that are able to change their structure at runtime, for instance when some exceptional events occur. In order to model the exception handling properly and allow structural changes of the system in a modular way, we introduce a new class of nets, called *adaptive workflow nets*. Adaptive workflow nets are a special type of Nets in Nets and they allow for creation, deletion and transformation of net tokens at runtime and for two types of synchronizations: synchronization on proper termination and synchronization on exception. We define some behavioral properties of adaptive workflow nets: *soundness* and *circumspectness* and employ an abstraction to reduce the verification of these properties to the verification of behavioral properties of a finite state abstraction.

Further, we study how formal methods can help in understanding and designing business processes. We investigate this for the extended event-driven process chains (eEPCs), a popular industrial business process language used in the ARIS Toolset. Several semantics have been proposed for EPCs. However, most of them concentrated solely on the control flow. We argue that other aspects of business processes must also be taken into account in order to analyze eEPCs and propose a semantics that takes data and time information from eEPCs into account. Moreover, we provide a translation of eEPCs to Timed Colored Petri nets in order to facilitate verification of eEPCs.

Finally, we discuss modeling issues for business processes whose behavior may depend on the previous behavior of the *process*, history which is recorded by workflow management systems as a log. To increase the precision of models with respect to modeling choices depending on the process history, we introduce history-dependent guards. The obtained business processes are called *history-dependent processes*. We introduce a logic, called *LogLogics* for the specification of guards based on a log of a current running process and give an evaluation

algorithm for such guards. Moreover, we show how these guards can be used in practice and define *LogLogics* patterns for properties that occur most commonly in practice.

Samenvatting

Dit proefschrift behandelt de verbetering van de huidige modeleer- en verificatie technieken voor complexe bedrijfsprocessen. Het doel van dit proefschrift is om verschillende aspecten van bedrijfsprocessen te bepalen en specifieke oplossingen te geven om de complexiteit van de processen te kunnen hanteren.

Voornamelijk behandelen we de verificatie van een beëindigingseigenschap voor workflow-netten, namelijk generalized soundness, die op een compositionele manier geverifiëerd kan worden. We beschrijven een procedure voor de beslisbaarheid van generalized soundness. De nieuwe beslissingsprocedure kan vaststellen of een workflow-net generalized sound is en geeft een tegenvoorbeeld als dit niet het geval is. Wij geven experimentele resultaten verkregen met een implementatie en een procedure voor de compositionele verificatie van grote workflows gebruik makend van reductieregels.

Daarnaast richten we ons op het modelleren en de verificatie van adaptieve workflow, een soort workflow die zijn eigen structuur tijdens de uitvoering kan veranderen, bijvoorbeeld wanneer uitzonderingen optreden. Om exceptieafhandeling te modelleren en om structurele veranderingen van het systeem mogelijk te maken op een modulaire manier, introduceren we een bijzonder soort netten, namelijk adaptieve netten. Adaptieve netten ondersteunen de aanmaak, verwijdering en transformatie van netten binnen netten tijdens de uitvoering en bovendien twee soorten synchronisaties: correcte beëindiging en exceptieafhandeling. We definiëren twee gedragseigenschappen van adaptieve workflow: soundness en circumspectness en we geven procedures voor de verificatie van deze twee eigenschappen, gebruik makend van abstractie.

Verder concentreren we ons op de toepassing van formele verificatiemethoden om informele bedrijfsprocessen te begrijpen en te ontwerpen. We onderzoeken extended event-driven process chains (eEPCs), een bedrijfsproces-taal gebruikt in de ARIS Toolset die veelvuldig in de industrie gebruikt wordt. Er zijn veel verschillende semantieken voor EPCs voorgesteld, maar de meeste daarvan beperken zich tot de volgordelijkheid en tot manieren om de semantiek daarvan te versterken om correcte processen te modelleren. We bepleiten dat bij de verificatie van eEPCs voor een correcte analyse met alle aspecten van het bedrijfsproces rekening gehouden moet worden en presenteren een semantiek die niet alleen volgorde maar ook data (resources) en tijd ondersteunt. Ook geven we een vertaling van eEPCs naar gekleurde (d.w.z. hoog-niveau) Petri-netten om de verificatie van eigenschappen met bestaande tools mogelijk te maken.

Uiteindelijk bespreken we het modelleren van bedrijfsprocessen waarvan het gedrag afhangt van eerder vertoond gedrag. De procesgeschiedenis wordt vaak al vastgelegd in een logbestand als een reeks gebeurtenissen met tijdsstempels. Het idee is nu om keuzes afhankelijk te maken van eigenschappen van de proces-

geschiedenis. We introduceren een logica, genaamd *LogLogics*, om zulke eigenschappen te formuleren voor onvolledige logs, dat wil zeggen, logs die alleen de procesgeschiedenis weergeven vanaf een bepaald tijdstip en we geven een algoritme voor hun waarheidsbepaling. Ook beschouwen we hoe zulke voorwaarden in de praktijk gebruikt kunnen worden en definiëren *LogLogics*-patronen voor eigenschappen die in de praktijk veel voor zullen komen.

Acknowledgments

All the work presented in this thesis was done under the thorough guidance of my supervisor Natalia Sidorova. She always gives sharp and practical advice regarding research. I would like to acknowledge Kees van Hee, my promotor, for his eagerness to get new results and for always sharing his practical insights on various topics of research. This has led to the variety of topics tackled in this thesis. I would like to thank both my supervisors for their support, especially in the last few months, which led to big improvements in the quality of the thesis.

I want to express my gratitude to the members of the reading committee: Wil van der Aalst, Irina Lomazova and Wolfgang Reisig. In particular, I thank Wil for his constructive comments which led to many corrections and presentation improvements, Wolfgang for helpful suggestions and Irina for fruitful discussions which led to our paper on adaptive nets. I am grateful to Wan Fokkink and Jos Baeten for accepting to act as opponents.

The work presented in this thesis was done in collaboration with the small Petri net group in the Architecture of Information System group. I would like to thank Alexander Serebrenik and Marc Voorhoeve for their enthusiasm, sharp criticism and sense of humor, which made working with them an exciting experience.

Many people helped in improving my work, of which some deserve a special mention. I want to thank Reinier Post for all his help throughout these years, especially in the tooling department, Jan Martijn van der Werf, for the nice collaboration during his master studies and for his patience during the last year as my office mate, Eric Verbeek for giving me the opportunity to work with ARIS Toolset and Natalia Ioustinova for her support in the initial stages of my PhD.

As a Beta PhD Student I was in frequent contact with the Information system group of Technology Management Department. This proved to be a valuable experience for me while trying to dig into the BPM research topics. Many thanks to the present and former members of the Information System department at the Department of Mathematics and Computer Science for creating a nice working atmosphere. I was always happy to have BEST discussions with my BEST colleagues from Berlin and Rostock and take part at inspiring LaQuSo lunches. Many thanks to Riet van Buul for her support throughout the PhD process.

Lots of thanks to Roxana Dietze for lots of useful discussions we have had along these years, in particular on our common “nets in nets” topic. Roxana was always able to cheer me up and bring me back on track, especially during the periods when Natalia was not available. Many thanks go to Simona Orzan for the discussions we had on the eEPC chapter and for all the support given

in the months leading to the completion of my thesis. I am grateful to Ferucio Laurentiu Tiplea, Ioana Leahu and to former colleagues in Iaşi, for their encouragements and help throughout these years. Last but not least, I would like to thank my parents for their constant belief in me.

October 2007, Eindhoven Olivia Oanea

Curriculum Vitae

Olivia Oanea was born on March 3, 1980 in Iaşi, Romania. In 1998 she graduated from the National College in Iaşi and started her studies at the Faculty of Computer Science (FII) of the "Al. I. Cuza" University of Iaşi. In 2001 she had a 5 months internship at the University of Konstanz, Germany as an Erasmus exchange student. In 2002 she received her BSc degree from FII with a thesis on decidability and complexity of Petri nets problems under the supervision of Prof. Dr. F. L. Tiplea. In October 2002 she started her Master studies on distributed systems at FII which she completed with a thesis on model checking extensions of Petri nets under the supervision of Prof. Dr. F. L. Tiplea. During her master studies, she also worked as a part time teaching assistant.

In July 2004 she started her PhD study at the Architecture of Information Systems Group, at the Department of Mathematics and Computer Science, Eindhoven University of Technology. Under the supervision of Dr. N. Sidorova and Prof. Dr. K.M. van Hee, she worked on various topics related to modeling and verification of business processes within the NWO project MoVeBP. Her research was focused on improving verification techniques for business processes modeled by (extensions of) Petri nets. This thesis contains the most important results of this work.