There and Back Again

On the Reconstructability and Rediscoverability of Typed Jackson Nets

Daniël Barenholz[1(✉)], Marco Montali[2], Artem Polyvyanyy[3], Hajo A. Reijers[1], Andrey Rivkin[2,4], and Jan Martijn E. M. van der Werf[1]

[1] Department of Information and Computing Sciences, Utrecht University, Princetonplein 5, 3584 CC Utrecht, The Netherlands
{d.barenholz,h.a.reijers,j.m.e.m.vanderwerf}@uu.nl
[2] Faculty of Computer Science, Free University of Bozen-Bolzano, piazza Domenicani 3, 39100 Bolzano, Italy
montali@inf.unibz.it
[3] The University of Melbourne, Melbourne, VIC 3010, Australia
artem.polyvyanyy@unimelb.edu.au
[4] Department of Applied Mathematics and Computer Science, Technical University of Denmark, Richard Petersens Plads 321, 2800 Kgs. Lyngby, Denmark
ariv@dtu.dk

Abstract. A process discovery algorithm aims to construct a model from data generated by historical system executions such that the model describes the system well. Consequently, one desired property of a process discovery algorithm is *rediscoverability*, which ensures that the algorithm can construct a model that is behaviorally equivalent to the original system. A system often simultaneously executes multiple processes that interact through object manipulations. This paper presents a framework for developing process discovery algorithms for constructing models that describe interacting processes based on typed Jackson Nets that use identifiers to refer to the objects they manipulate. Typed Jackson Nets enjoy the *reconstructability* property which states that the composition of the processes and the interactions of a decomposed typed Jackson Net yields a model that is bisimilar to the original system. We exploit this property to demonstrate that if a process discovery algorithm ensures rediscoverability, the system of interacting processes is rediscoverable.

1 Introduction

Business processes are fundamental to a wide range of systems. A business process is a collection of activities that, when performed, aims to achieve a business objective at an organization. Examples of business processes are an order-to-cash process at a retailer, a medical assessment process at a hospital, or a credit check process at a bank. Business processes are modeled using process modeling languages, such as Petri nets, and used for communication and analysis purposes [1]. Petri nets provide a graphical representation of the flow of activities within a

L. Gomes and R. Lorenz (Eds.): PETRI NETS 2023, LNCS 13929, pp. 37–58, 2023.
https://doi.org/10.1007/978-3-031-33620-1_3

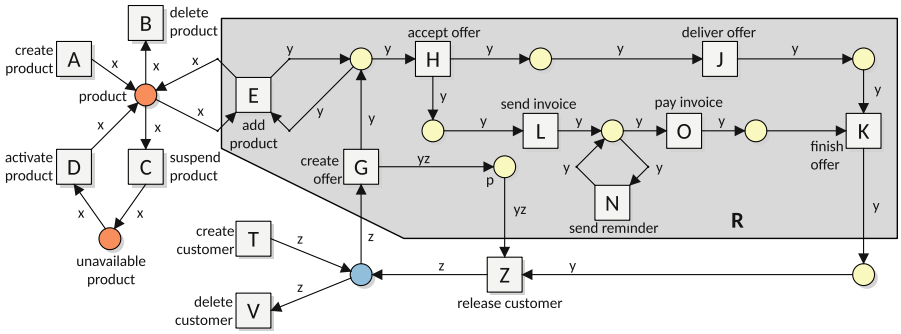


Fig. 1. A retailer system of three interacting processes.

process and can be used to model various types of concurrent and sequential behavior [18].

A process discovery algorithm aims to automatically construct a model from data generated by historical process executions captured in an event log of the system, such that the model describes the system well. A desired property of a discovery algorithm is *rediscoverability*. This property states that if a system S, expressed as a model M, generates an event log L, then a discovery algorithm with the rediscoverability property should construct M from L. In other words, the algorithm can reverse engineer the model of the system from the data the model has generated. Only a few existing algorithms guarantee this property. For example, if the model is a block-structured workflow net, and the event log is directly-follows complete, then the α-Miner algorithm [22] can rediscover the net that generated the event log. Similarly, again under the assumption that the event log is directly-follows complete, Inductive Miner [16] can rediscover process trees without duplicate transitions, self-loops, or silent transitions.

Most existing process discovery algorithms assume that a system executes a single process [4]. Consequently, an event log is defined as a collection of sequences where a sequence describes the execution of a single process instance. However, many information systems, such as enterprise resource planning systems, do not satisfy this assumption. A system often executes multiple interacting processes [11,23]. For example, consider a retailer system that executes three processes: an order, product, and customer management process, as depicted in Fig. 1. These processes are intertwined. Specifically, only available products may be ordered, and customers can only have one order at a time. Consequently, events do not belong to a single process but relate to several processes. For instance, consider an event e in some event log that occurred as transition G was executed for some customer c and created a new order o in the system. Event e relates to the customer process instance c and the order process instance o. Traditional process discovery techniques require event e to be stored in multiple event logs and generate multiple models, one for each process [7].

A different approach is taken in artifact or object-centric process discovery [5,17] and agent system discovery [20,21]. In object-centric process

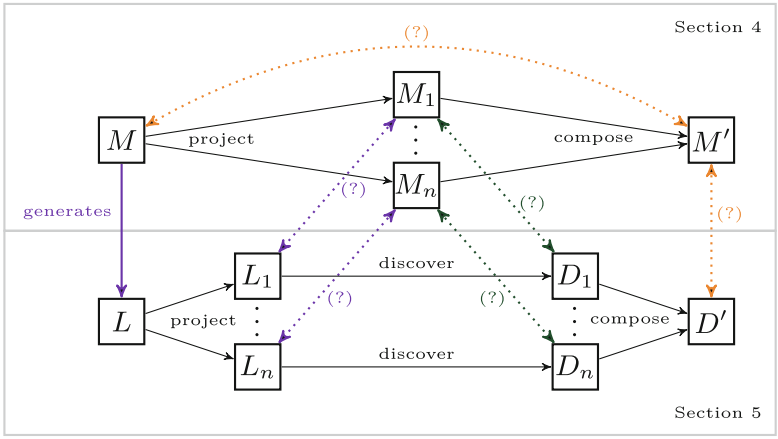


Fig. 2. The framework for rediscoverability of systems of interacting processes.

discovery, instead of linking each event to a single object, events can be linked to multiple objects stored in object-centric event logs [9]. Existing object-centric discovery algorithms project the input event log on each object type to create a set of "flattened" event logs. For each event log, a model is discovered, after which these models are combined into a single model [5]. In general, flattening is lossy [7], as in this step, events can disappear [5], be duplicated (convergence) [3], or lead to wrong event orders (divergence) [3]. In agent system discovery, instead of interacting objects, a system is viewed as composed of multiple autonomous agents, each driving its processes that interact to achieve an overall objective of the system [20]. An agent system discovery algorithm proceeds by decomposing the input event log into multiple event logs, each composed of events performed by one agent (type) and an event log of interactions, and then discovering agent and interaction models and composing them into the resulting system [21].

In this paper, we study under what conditions projections in event logs can guarantee rediscoverability for interacting processes, represented as typed Jackson Nets, a subclass of typed Petri nets with identifiers [19,23]. The class of typed Jackson Nets is inspired by Box Algebra [10] and Jackson Nets [14], which are (representations of) block-structured workflow nets that are *sound* [2] by construction [16]. As we demonstrate, typed Jackson Nets exhibit a special property: they are *reconstructable*. Composing the projections of each type is insufficient for reconstructing a typed Jackson Net. Instead, if the subset-closed set of all type combinations is considered, the composition returns the original model of the system. We show how the reconstructability property can be used to develop a framework for rediscoverability of typed Jackson Nets using traditional process discovery algorithms. The framework builds upon a divide and conquer strategy, as depicted in Fig. 2. The principle idea of this strategy is to project an event log L generated by some model M of the system onto logs $L_1, \ldots, L_n$. Then, if these projected event logs satisfy the conditions of a process discovery algorithm, composition of the resulting models $D_1, \ldots, D_n$ into model D' should

rediscover the original model of the system. In this framework, we show that every projected event log is also an event log of the corresponding projected model. Consequently, if a process discovery algorithm guarantees the rediscoverability of projected models, then the composition operator for typed Jackson Nets can be used to ensure the rediscoverability of the original system.

The next section presents the basic notions. In Sect. 3, we introduce typed Jackson Nets, which, as shown in Sect. 4, are reconstructable. We define a framework for developing discovery algorithms that guarantee rediscoverability in Sect. 5. We conclude the paper in Sect. 6. Full proofs of the lemmata and theorems can be found in [8].

2 Preliminaries

Let S and T be two possibly infinite sets. The powerset of S is denoted by $\mathcal{P}(S) = \{S' \mid S' \subseteq S\}$ and $|S|$ denotes the cardinality of S. Two sets S and T are *disjoint* if $S \cap T = \emptyset$, with $\emptyset$ denoting the empty set. The cartesian product of two sets S and T, is defined by $S \times T = \{(a, b) \mid a \in S, b \in T\}$. The generalized cartesian product for some set S and and sets T_s for $s \in S$ is defined as $\Pi_{s \in S} T_s = \left\{f : S \to \bigcup_{s \in S} T_s \mid \forall s \in S : f(s) \in T_s\right\}$. Given a relation $R \subseteq S \times T$, its range is defined by $\textsc{rng}(R) = \{y \in T \mid \exists x \in S : (x, y) \in R\}$. Similarly, the domain of R is defined by $\textsc{dom}(R) = \{x \in S \mid \exists y \in T : (x, y) \in R\}$. Restricting the domain of a relation to a set U is defined by $R_{|U} = \{(a, b) \in R \mid a \in U\}$.

A *multiset* m over S is a mapping of the form $m : S \to \mathbb{N}$, where $\mathbb{N} = \{0, 1, 2, \ldots\}$ denotes the set of natural numbers. For $s \in S$, $m(s) \in \mathbb{N}$ denotes the number of times s appears in multiset m. We write s^n if $m(s) = n$. For $x \notin S$, $m(x) = 0$. We use $S^{\oplus}$ to denote the set of all finite multisets over S and overload $\emptyset$ to also denote the empty multiset. The size of a multiset is defined by $|m| = \sum_{s \in S} m(s)$. The support of $m \in S^{\oplus}$ is the set of elements that appear in m at least once: $\mathit{supp}\,(m) = \{s \in S \mid m(s) > 0\}$. Given two multisets m_1 and m_2 over S: *(i)* $m_1 \subseteq m_2$ iff $m_1(s) \leq m_2(s)$ for each $s \in S$; *(ii)* $(m_1 + m_2)(s) = m_1(s) + m_2(s)$ for each $s \in S$; and *(iii)* if $m_1 \subseteq m_2$, $(m_2 - m_1)(s) = m_2(s) - m_1(s)$ for each $s \in S$.

A *sequence* over S of length $n \in \mathbb{N}$ is a function $\sigma : \{1, \ldots, n\} \to S$. If $n > 0$ and $\sigma(i) = a_i$, for $1 \leq i \leq n$, we write $\sigma = \langle a_1, \ldots, a_n \rangle$. The length of a sequence σ is denoted by $|\sigma|$. The sequence of length 0 is called the *empty sequence*, and is denoted by ϵ. The set of all finite sequences over S is denoted by S^*. We write $a \in \sigma$ if there is $1 \leq i \leq |\sigma|$ such that $\sigma(i) = a$ and $\mathit{supp}\,(\sigma) = \{a \in S \mid \exists 1 \leq i \leq |\sigma| : \sigma(i) = a\}$. *Concatenation* of two sequences $\nu, \gamma \in S^*$, denoted by $\sigma = \nu \cdot \gamma$, is a sequence defined by $\sigma : \{1, \ldots, |\nu| + |\gamma|\} \to S$, such that $\sigma(i) = \nu(i)$ for $1 \leq i \leq |\nu|$, and $\sigma(i) = \gamma(i - |\nu|)$ for $|\nu| + 1 \leq i \leq |\nu| + |\gamma|$. Projection of sequences on a set T is defined inductively by $\epsilon_{|T} = \epsilon$, $(\langle a \rangle \cdot \sigma)_{|T} = \langle a \rangle \cdot \sigma_{|T}$ if $a \in T$ and $(\langle a \rangle \cdot \sigma)_{|T} = \sigma_{|T}$ otherwise. Renaming a sequence with an injective function $r : S \to T$ is defined inductively by $\rho_r(\epsilon) = \epsilon$, and $\rho_r(\langle a \rangle \cdot \sigma) = \langle r(a) \rangle \cdot \rho_r(\sigma)$. Renaming is extended to multisets of sequences as follows: given a

multiset $m \in (S^*)^{\oplus}$, we define $\rho_r(m) = \sum_{\sigma \in supp(m)} \sigma(m) \cdot \rho_r(\sigma)$. For example, $\rho_{\{x \mapsto a, y \mapsto b\}}(\langle x, y \rangle^3) = \langle a, b \rangle^3$.

A *directed graph* is a pair (V, A) where V is the set of vertices, and $A \subseteq V \times V$ the set of arcs. Two graphs $G_1 = (V_1, A_1)$ and $G_2 = (V_2, A_2)$ are *isomorphic*, denoted by $G_1 \leftrightsquigarrow G_2$, if a bijection $b : V_1 \to V_2$ exists, such that $(v_1, v_2) \in A_1$ iff $(b(v_1), b(v_2)) \in A_2$.

Given a finite set A of (action) labels, a *(labeled) transition system* (LTS) over A is a tuple $\Gamma_A = (S, A, s_0, \rightarrow)$, where S is the (possibly infinite) set of *states*, s_0 is the *initial state* and $\rightarrow \subset (S \times (A \cup \{\tau\}) \times S)$ is the *transition relation*, where $\tau \notin A$ denotes the silent action [13]. In what follows, we write $s \xrightarrow{a} s'$ for $(s, a, s') \in \rightarrow$. Let $r : A \to (A' \cup \{\tau\})$ be an injective, total function. Renaming Γ with r is defined as $\rho_r(\Gamma) = (S, A \setminus A', s_0, \rightarrow')$ with $(s, r(a), s') \in \rightarrow'$ iff $(s, a, s') \in \rightarrow$. Given a set T, hiding is defined as $\hat{\mathtt{H}}_T(\Gamma) = \rho_h(\Gamma)$ with $h : A \to A \cup \{\tau\}$ such that $h(t) = \tau$ if $t \in T$ and $h(t) = t$ otherwise. Given $a \in A$, $p \stackrel{a}{\Rightarrow} q$ denotes a *weak transition relation* that is defined as follows: *(i)* $p \stackrel{a}{\Rightarrow} q$ iff $p(\xrightarrow{\tau})^* q_1 \xrightarrow{a} q_2(\xrightarrow{\tau})^* q$; *(ii)* $p \stackrel{\tau}{\Rightarrow} q$ iff $p(\xrightarrow{\tau})^* q$. Here, $(\xrightarrow{\tau})^*$ denotes the reflexive and transitive closure of $\xrightarrow{\tau}$.

Let $\Gamma_1 = (S_1, A, s_{01}, \rightarrow_1)$ and $\Gamma_2 = (S_2, A, s_{02}, \rightarrow_2)$ be two LTSs. A relation $R \subseteq (S_1 \times S_2)$ is called a *strong simulation*, denoted as $\Gamma_1 \prec_R \Gamma_2$, if for every pair $(p, q) \in R$ and $a \in A \cup \{\tau\}$, it holds that if $p \xrightarrow{a}_1 p'$, then there exists $q' \in S_2$ such that $q \xrightarrow{a}_2 q'$ and $(p', q') \in R$. Relation R is a *weak simulation*, denoted by $\Gamma_1 \preccurlyeq_R \Gamma_2$, iff for every pair $(p, q) \in R$ and $a \in A \cup \{\tau\}$ it holds that if $p \xrightarrow{a}_1 p'$, then $a = \tau$ and $(p', q) \in R$, or there exists $q' \in S_2$ such that $q \stackrel{a}{\Rightarrow}_2 q'$ and $(p', q') \in R$. Relation R is called a strong (weak) *bisimulation*, denoted by $\Gamma_1 \sim_R \Gamma_2$ ($\Gamma_1 \approx_R \Gamma_2$) if both $\Gamma_1 \prec \Gamma_2$ ($\Gamma_1 \preccurlyeq_R \Gamma_2$) and $\Gamma_2 \prec_{R^{-1}} \Gamma_1$ ($\Gamma_2 \preccurlyeq_{R^{-1}} \Gamma_1$). Given a strong (weak) (bi)simulation R, we say that a state $p \in S_1$ is strongly (weakly) rooted (bi)similar to $q \in S_2$, written $p \sim^r_R q$ (correspondingly, $p \approx^r_R q$), if $(p, q) \in R$. The relation is called *rooted* iff $(s_{01}, s_{02}) \in R$. A rooted relation is indicated with a superscript ^{r}.

A weighted Petri net is a 4-tuple (P, T, F, W) where P and T are two disjoint sets of *places* and *transitions*, respectively, $F \subseteq ((P \times T) \cup (T \times P))$ is the *flow relation*, and $W : F \to \mathbb{N}^+$ is a *weight function*. For $x \in P \cup T$, we write ${}^\bullet x = \{y \mid (y, x) \in F\}$ to denote the *preset* of x and $x^\bullet = \{y \mid (x, y) \in F\}$ to denote the *postset* of x. We lift the notation of preset and postset to sets element-wise. If for a Petri net no weight function is defined, we assume $W(f) = 1$ for all $f \in F$. A *marking* of N is a multiset $m \in P^{\oplus}$, where $m(p)$ denotes the number of *tokens* in place $p \in P$. If $m(p) > 0$, place p is called *marked* in marking m. A *marked Petri net* is a tuple (N, m) with N a weighted Petri net with marking m. A transition $t \in T$ is enabled in (N, m), denoted by $(N, m)\,[t\rangle$ iff $W((p, t)) \leq m(p)$ for all $p \in {}^\bullet t$. An enabled transition can *fire*, resulting in marking m' iff $m'(p) + W((p, t)) = m(p) + W((t, p))$, for all $p \in P$, and is denoted by $(N, m)\,[t\rangle\,(N, m')$. We lift the notation of firings to sequences. A sequence $\sigma \in T^*$ is a *firing sequence* iff $\sigma = \epsilon$, or markings $m_0, \ldots, m_n$ exist such that $(N, m_{i-1})[\sigma(i)\rangle(N, m_i)$ for $1 \leq i \leq |\sigma| = n$, and is denoted by $(N, m_0)[\sigma\rangle(N, m_n)$. If the context is clear, we omit the weighted Petri net

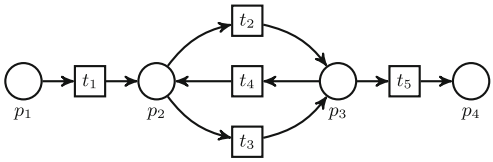


Fig. 3. An example block-structured WF-net. Each block corresponds to a node in the Jackson type $(p_1; (t_1; (((p_2; ((t_2 + t_3)\,; p_3))\,\#t_4)\,; (t_5; p_4))))$. As example, the choice between transitions t_2 and t_3 corresponds to the node $(p_2; ((t_2 + t_3)\,; p_3))$.

N. The set of reachable markings of (N, m) is defined by $\mathcal{R}(N, m) = \{m' \mid \exists \sigma \in T^* : m[\sigma\rangle m'\}$. The set of all possible finite firing sequences of (N, m) is denoted by $\mathcal{L}(N, m_0) = \{\sigma \in T^* \mid m[\sigma\rangle m'\}$. The semantics of a marked Petri net (N, m) with $N = (P, T, F, W)$ is defined by the LTS $\Gamma_{N,m} = (P^{\oplus}, T, m_0, \rightarrow)$ with $(m, t, m') \in \rightarrow$ iff $m[t\rangle m'$. A Petri net $N = (P, T, F, W)$ has underlying graph $(P \cup T, F)$. Two Petri nets N and N' are isomorphic, denoted using $N \leftrightsquigarrow N'$, if their underlying graphs are.

A *workflow net* (WF-net for short) is a tuple $N = (P, T, F, W, in, out)$ such that: *(i)* (P, T, F, W) is a weighted Petri net; *(ii)* $in, out \in P$ are the source and sink place, respectively, with ${}^\bullet in = out^\bullet = \emptyset$; *(iii)* every node in $P \cup T$ is on a directed path from in to out. N is called *k-sound* for some $k \in \mathbb{N}$ iff *(i)* it is proper completing, i.e., for all reachable markings $m \in \mathcal{R}(N, [in^k])$, if $[out^k] \subseteq m$, then $m = [out^k]$; *(ii)* it is weakly terminating, i.e., for any reachable marking $m \in \mathcal{R}(N, [in^k])$, the final marking is reachable, i.e., $[out^k] \in \mathcal{R}(N, m)$; and *(iii)* it is quasi-live, i.e., for all transitions $t \in T$, there is a marking $m \in \mathcal{R}(N, [in])$ such that $m[t\rangle$. The net is called *sound* if it is 1-sound.

3 Typed Jackson Nets to Model Interacting Processes

In this section, we introduce typed Jackson Nets as subclass of typed Petri nets with identifiers. We show that this class is a natural extension to Jackson Nets, which are representations of block-structured workflow nets. Typed Jackson Nets are identifier sound and live by construction.

3.1 Jackson Nets

Whereas WF-nets do not put any restriction on the control flow of activities, block-structured WF-nets divide the control flow in logical blocks [15]. Each "block" represents a single unit of work that can be performed, where this unit of work is either atomic (single transition), or one involving multiple steps (multiple transitions). An example block-structured WF-net is shown in Fig. 3. The main advantage of block-structured WF-nets, is that the block-structure ensures that the WF-net is sound by definition [14–16]. In this paper, we consider Jackson Types and Jackson Nets [14]. A Jackson Type is a data structure used to capture all information involved in a single execution of a WF-net.

Definition 1 (Jackson Type [14]). *The set of* Jackson Types $\mathcal{J}$ *is recursively defined by the following grammar:*

$$\begin{aligned}\mathcal{J} &::= \mathscr{A}^p \mid (\mathscr{A}^p; (\mathcal{J}^t; \mathscr{A}^p)) \\ \mathcal{J}^t &::= \mathscr{A}^t \mid (\mathcal{J}^t; (\mathcal{J}^p; \mathcal{J}^t)) \mid (\mathcal{J}^t + \mathcal{J}^t) \\ \mathcal{J}^p &::= \mathscr{A}^p \mid (\mathcal{J}^p; (\mathcal{J}^t; \mathcal{J}^p)) \mid (\mathcal{J}^p \parallel \mathcal{J}^p) \mid (\mathcal{J}^p \# \mathcal{J}^t)\end{aligned}$$

where $\mathscr{A} = \mathscr{A}^p \cup \mathscr{A}^t = \{a, b, c, \ldots\}$ *denotes two disjoint sets of atomic types for places and transitions, resp., and symbols* $;, \parallel, +, \#$ *stand for sequence, parallelism, choices, and loops.* ◁

A Jackson Net is a Petri net where each place has an assigned Jackson Type. The class of Jackson Nets is obtained by recursively applying *generation rules*, starting from a singleton net with only one place. These generation rules are similar to those defined by Murata [18] and preserve soundness [14]. Thus, any Jackson Net is sound by construction.

Definition 2 (Jackson Net [14]). *A WF-net* $N = (P, T, F, in, out)$ *is called a* Jackson Net *if it can be generated from a single place* p *by applying the following five generation rules recursively:*

$$\begin{array}{ll} J1\text{:}\ p \leftrightarrow (p_1; (t; p_2)) & J4\text{:}\ p \leftrightarrow (p_1 \parallel p_2) \\ J2\text{:}\ t \leftrightarrow (t_1; (p_1; t_2)) & J5\text{:}\ t \leftrightarrow (t_1 + t_2) \\ J3\text{:}\ p \leftrightarrow (p \# t) & \end{array}$$

We say that N *is generated by* p. ◁

As shown in [14], Jackson Nets are completely determined by Jackson Types, and vice versa.

3.2 Petri Nets with Identifiers

Whereas WF-nets describe all possible executions for a single case, systems typically consist of many interacting processes. The latter can be modeled using typed Petri nets with identifiers (t-PNIDs for short) [23]. In this formalism, each object is typed and has a unique identifier to be able to refer to it. Tokens carry vectors of identifiers, which are used to relate objects. Variables on the arcs are used to manipulate the identifiers.

Definition 3 (Identifiers, Types and Variables). *Let* $\mathcal{I}$ *,* Λ*, and* $\mathcal{V}$ *denote countably infinite sets of identifiers, type labels, and variables, respectively. We define:*

- *the* domain assignment *function* $I : \Lambda \to \mathcal{P}(\mathcal{I})$*, such that* $I(\lambda_1)$ *is an infinite set, and* $I(\lambda_1) \cap I(\lambda_2) \neq \emptyset$ *implies* $\lambda_1 = \lambda_2$ *for all* $\lambda_1, \lambda_2 \in \Lambda$*;*
- *the* id typing *function* $\mathtt{type}_{\mathcal{I}} : \mathcal{I} \to \Lambda$ *s.t. if* $\mathtt{type}_{\mathcal{I}}(\mathtt{id}) = \lambda$*, then* $\mathtt{id} \in I(\lambda)$*;*
- *a* variable typing *function* $\mathtt{type}_{\mathcal{V}} : \mathcal{V} \to \Lambda$*, prescribing that* $x \in \mathcal{V}$ *can be substituted only by values from* $I(\mathtt{type}_{\mathcal{V}}(x))$*.*

When clear from the context, we omit the subscripts of $\texttt{type}$. *We lift the* $\texttt{type}$ *functions to sets, vectors, and sequences by applying the function on each of its constituents.* ◁

In a t-PNID, each place is annotated with a label, called the *place type*. A place type is a vector of types, indicating types of identifier tokens the place can carry. Similar to Jackson Types, we use $[p, \lambda]$ to denote that place p has type $\alpha(p) = \lambda$. Each arc is inscribed with a multiset of vectors of identifiers, such that the type of each variable coincides with the place types. If the inscription is empty or contains a single element, we omit the brackets.

Definition 4 (Typed Petri net with identifiers). *A* typed Petri net with identifiers *(t-PNID)* N *is a tuple* (P, T, F, α, β), *where:*

- (P, T, F) *is a classical Petri net;*
- $\alpha : P \to \Lambda^*$ *is the* place typing function*;*
- $\beta : F \to (\mathcal{V}^*)^\oplus$ *defines for each arc a multiset of* variable vectors *s.t.* $\alpha(p) = \texttt{type}(x)$ *for any* $x \in supp\,(\beta((p,t)))$ *and* $\texttt{type}(y) = \alpha(p')$ *for any* $y \in supp\,(\beta((t,p')))$ *where* $t \in T$, $p \in {}^\bullet t$, $p' \in t^\bullet$. ◁

A marking of a t-PNID is the configuration of tokens over the set of places. Each token in a place should be of the correct type, i.e., the vector of identifiers carried by a token in a place should match the corresponding place type. The set $\texttt{C}(p)$ defines all possible vectors of identifiers a place p may carry.

Definition 5 (Marking). *Given a t-PNID* $N = (P, T, F, \alpha, \beta)$, *and place* $p \in P$, *its* id set *is* $\texttt{C}(p) = \prod_{1 \le i \le |\alpha(p)|} I(\alpha(p)(i))$. *A* marking *is a function* $m \in \mathbb{M}(N)$, *with* $\mathbb{M}(N) = P \to (\Lambda^*)^\oplus$, *such that* $m(p) \in \texttt{C}(p)^\oplus$, *for each place* $p \in P$. *The set of identifiers used in* m *is denoted by* $Id(m) = \bigcup_{p \in P} \text{RNG}(supp\,(m(p)))$ *The pair* (N, m) *is called a* marked t-PNID. ◁

To define the semantics of a t-PNID, the variables need to be valuated with identifiers.

Definition 6 (Variable sets [23]). *Given a t-PNID* $N = (P, T, F, \alpha, \beta)$, $t \in T$ *and* $\lambda \in \Lambda$, *we define the following sets of variables:*

- input variables *as* $In(t) = \bigcup_{x \in \beta((p,t)), p \in {}^\bullet t} \text{RNG}(supp\,(x))$;
- output variables *as* $Out(t) = \bigcup_{x \in \beta((t,p)), p \in t^\bullet} \text{RNG}(supp\,(x))$;
- variables *as* $Var(t) = In(t) \cup Out(t)$;
- emitting variables *as* $Emit(t) = Out(t) \setminus In(t)$;
- collecting variables *as* $Collect(t) = In(t) \setminus Out(t)$;
- emitting transitions *as* $E_N(\lambda) = \{t \mid \exists x \in Emit(t) \wedge \texttt{type}(x) = \lambda\}$;
- collecting transitions *as* $C_N(\lambda) = \{t \mid \exists x \in Collect(t) \wedge \texttt{type}(x) = \lambda\}$;
- types in N *as* $\texttt{type}(N) = \{\vec{\lambda} \mid \exists p \in P : \vec{\lambda} \in \alpha(p)\}$. ◁

A valuation of variables to identifiers is called a *binding*. Bindings are used to inject new fresh data into the net via variables that emit identifiers, i.e., via variables that appear only on the output arcs of that transition. Note that in this definition, freshness of identifiers is local to the marking, i.e., disappeared

identifiers (those fully removed from the net through collecting transitions) may be reused, as it does not hamper the semantics of the t-PNID.

Definition 7 (Firing rule for t-PNIDs). *Given a marked t-PNID* (N, m) *with* $N = (P, T, F, \alpha, \beta)$*, a* binding *for transition* $t \in T$ *is an injective function* $\psi : \mathcal{V} \to \mathcal{I}$ *such that* $\texttt{type}(v) = \texttt{type}(\psi(v))$ *and* $\psi(v) \notin Id(m)$ *iff* $v \in Emit(t)$*. Transition* t *is* enabled *in* (N, m) *under binding* ψ*, denoted by* $(N, m)[t, \psi\rangle$ *iff* $\rho_\psi(\beta(p, t)) \leq m(p)$ *for all* $p \in {}^\bullet t$*. Its firing results in marking* m'*, denoted by* $(N, m)[t, \psi\rangle(N, m')$*, such that* $m'(p) + \rho_\psi(\beta(p, t)) = m(p) + \rho_\psi(\beta(t, p))$*.* ◁

The firing rule is inductively extended to sequences. A marking m' is *reachable* from m if there exists $\eta \in (T \times (\mathcal{V} \to \mathcal{I}))^*$ such that $(N, m)[\eta\rangle(N, m')$. We denote with $\mathcal{R}(N, m)$ the set of all markings reachable from m for (N, m). We use $\mathcal{L}(N, m)$ to denote all possible firing sequences of (N, m), i.e., $\mathcal{L}(N, m) = \{\eta \mid (N, m)[\eta\rangle\}$ and $Id(\eta) = \bigcup_{(t,\psi)\in\eta} \text{RNG}(\psi)$ for the set of identifiers used in η. The execution semantics of a t-PNID is defined as an LTS that accounts for all possible executions starting from a given initial marking. We say two t-PNIDs are bisimilar if their induced transition systems are.

Definition 8. *Given a marked t-PNID* (N, m_0) *with* $N = (P, T, F, \alpha, \beta)$*, its induced transition system is* $\Gamma_{N,m_0} = (\mathbb{M}(N), (T \times (\mathcal{V} \to \mathcal{I})), m_0, \to)$ *with* $m \xrightarrow{(t,\psi)} m'$ *iff* $(N, m)\,[t, \psi\rangle\,(N, m')$*.* ◁

Soundness properties for WF-nets typically consist of proper completion, weak termination, and quasi-liveness [6]. Extending soundness to t-PNIDs gives *identifier soundness* [23]. In t-PNIDs, each object of a given type "enters" the system through an emitting transition, binding it to a unique identifier. Identifier soundness intuitively states that it should always be possible to remove objects (weak type termination), and that once a collecting transition fires for an object, there should be no remaining tokens referring to the removed object (proper type completion).

Definition 9 (Identifier Soundness [23]). *Let* (N, m_0) *a marked t-PNID and* $\lambda \in \Lambda$ *some type.* (N, m_0) *is* λ-sound *iff it is*

- *Proper* λ*-completing, i.e., for all* $t \in C_N(\lambda)$*, bindings* $\psi : \mathcal{V} \to \mathcal{I}$ *and markings* $m, m' \in \mathcal{R}(N, m_0)$*, if* $m[t, \psi\rangle m'$*, then for all identifiers* $\texttt{id} \in \text{RNG}(\psi|_{Collect(t)}) \cap Id(m)$ *and* $\texttt{type}(\texttt{id}) = \lambda$*, it holds that* $\texttt{id} \notin Id(m')$[1]*;*
- *Weakly* λ*-terminating, i.e., for every* $m \in \mathcal{R}(N, m_0)$ *and identifier* $\texttt{id} \in I(\lambda)$ *such that* $\texttt{id} \in Id(m)$*, there exists a marking* $m' \in \mathcal{R}(N, m)$ *with* $\texttt{id} \notin Id(m')$*.*

If it is λ*-sound for all* $\lambda \in \texttt{type}(N)$*, then it is* identifier sound. ◁

3.3 Typed Jackson Nets

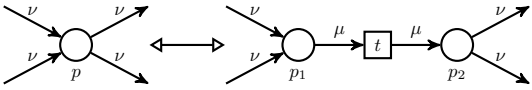
In general, identifier soundness is undecidable for t-PNIDs [23]. Similar as Jackson Nets restrict WF-nets to blocks, *typed Jackson Nets* (t-JNs) restrict t-PNIDs

[1] Here, we constrain ψ only to objects of type λ that are only consumed.

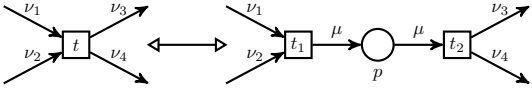
to blocks, while guaranteeing identifier soundness and liveness. For t-JNs, we disallow multiplicity on arcs and variables, i.e., $\beta(f)(v) \leq 1$ for all $f \in F$ and $v \in \mathcal{V}$, and imply a bijection on variables and identifier types. This prevents place types like $\lambda = \langle x, x \rangle$. Assuming a Gödel-like number on types (cf. [14]), place types and arc inscriptions can be represented as sets. Similar as Jackson Types describe Jackson Nets, we apply a notation based on Jackson Types to denote typed Jackson Nets.

Definition 10 (Typed Jackson Net). *A t-PNID N is a* typed Jackson Net *if it can be generated from a set of transitions T' by applying any of the following six generation rules recursively. If N is generated from a singleton set of transitions (i.e., $|T'| = 1$), N is called* atomic.

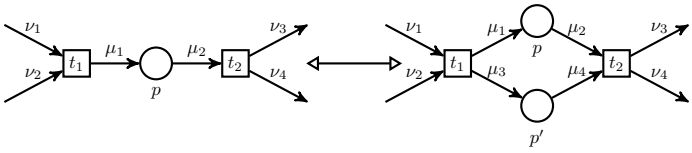
R1 Place Expansion: $[p, \lambda] \leftrightarrow ([p_1, \lambda]\,;(t_1;[p_2, \lambda]))$



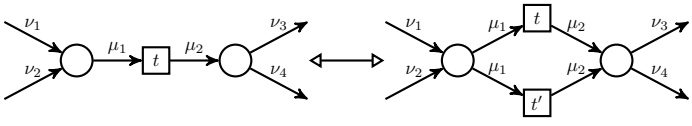
R2 Transition Expansion: $t \leftrightarrow (t_1; ([p, \lambda]\,; t_2))$*, with* $Var(t) \subseteq \lambda$



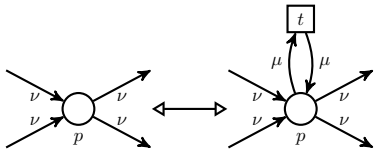
R3 Place Duplication: $(t_1; ([p, \lambda]\,; t_2)) \leftrightarrow (t_1; (([p, \lambda] \parallel [p', \lambda'])\,; t_2))$*,*
with $\lambda' \cap Emit(p^\bullet) = \emptyset$



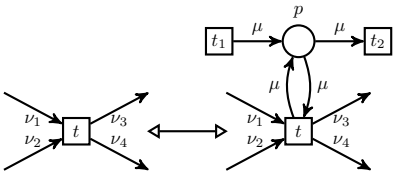
R4 Transition Duplication: $t \leftrightarrow (t + t')$



R5 Self Loop Addition: $[p, \lambda] \leftrightarrow ([p, \lambda]\,\#t)$



R6 Identifier Introduction: $t \leftrightarrow (t \triangleleft (N_1, [p, \lambda], N_2))$, *with* $(N_1; ([p, \lambda]; N_2))$ *a t-JN and* $\lambda \cap Var(t) = \emptyset$



◁

An example t-JN is given in Fig. 1. Starting with the product process, transitions C and D can be reduced using rule $R2$. The resulting transition is a self-loop transition, and can be reduced using $R5$, resulting in the block $(E \triangleleft (A, product, B))$. This block can be reduced using $R6$, leaving transition E. Transition E is again a self-loop, and can be reduced using $R5$. The block containing transitions H, J, L O, N and K can be reduced to a single place by applying rules $R1$, $R2$ and $R5$ repeatedly. The remaining place is a duplicate place with respect to place p, and can be reduced using $R3$. Applying $R2$ on G and Z results in the block $(G \triangleleft (T, customer, V))$, which can be reduced to the transition G. Hence, the net in Fig. 1 is an atomic t-JN.

Theorem 1 (Identifier Soundness of typed Jackson Nets [23]). *Given a t-JN* N*, then* N *is* identifier sound *and* live. ◁

4 Decomposability of t-JNs

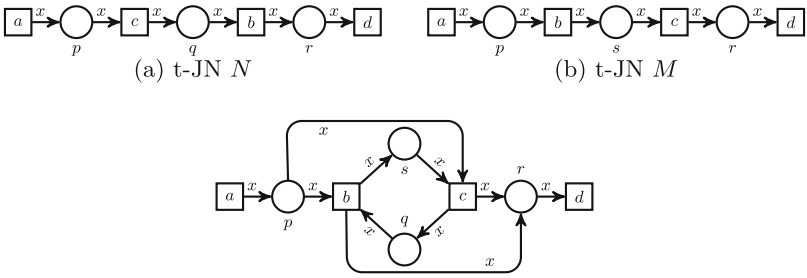
t-PNIDs specify a class of nets with explicitly defined interactions between objects of different types within one system. However, sometimes one may want to focus only on some behaviors exhibited by a given set of object types, by extracting a corresponding net from the original t-PNID model. We formalize this idea below.

Definition 11 (Type projection). *Let* $N = (P_N, T_N, F_N, \alpha, \beta)$ *be a t-PNID and* $\Upsilon \subseteq \Lambda$ *be a set of identifier types. The* type projection *of* Υ *on* N *is a t-PNID* $\pi_\Upsilon(N) = (P_\Upsilon, T_\Upsilon, F_\Upsilon, \alpha_\Upsilon, \beta_\Upsilon)$*, where:*

- $P_\Upsilon = \{p \in P \mid \Upsilon \subseteq \alpha(p)\}$;
- $T_\Upsilon = \{t \in T \mid ({}^\bullet t \cup t^\bullet) \cap P_\Upsilon \neq \emptyset\}$;
- $F_\Upsilon = F \cap ((P_\Upsilon \times T_\Upsilon) \cup (T_\Upsilon \times P_\Upsilon))$;
- $\alpha_\Upsilon(p) = \Upsilon$, *for each* $p \in P_\Upsilon$;
- $\beta_\Upsilon(f) = \beta(f)|_{\mathtt{type}_\mathcal{V}^{-1}(\Upsilon)}$, *for each* $f \in ((P_\Upsilon \times T_\Upsilon) \cup (T_\Upsilon \times P_\Upsilon))$. ◁

With the next lemma we explore a property of typed Jackson nets that, in a nutshell, shows that t-JNs are closed under the type projection. This also indirectly witnesses that t-JNs provide a suitable formalism for specifying and manipulating systems with multiple communicating components.

Lemma 1. *If* $N = (P_N, T_N, F_N, \alpha, \beta)$ *is a t-JN, then* $\pi_\Upsilon(N)$ *is a t-JN as well, for any* $\Upsilon \subseteq \mathtt{type}_\Lambda(N)$. ◁



(a) t-JN N (b) t-JN M

(c) t-PNID $N \uplus M$

Fig. 4. Although both N and M are t-JNs, their composition is not

Proof. (sketch) Let us assume for simplicity that N is atomic. Then, using rules from Definition 10, N can be reduced to a single transition. Starting from this transition, one can construct a t-JN following the net graph construction from Definition 11 using the same rules (but the identifier introduction one), proviso that arc inscriptions are always of type Υ. Then, it is easy to check that the constructed net is indeed the type projection of Υ on N. ■

We define next how t-PNIDs can be composed and show that t-JNs are not closed under the composition.

Definition 12 (Composition). *Let* $N = (P_N, T_N, F_N, \alpha_N, \beta_N)$ *and* $M = (P_M, T_M, F_M, \alpha_M, \beta_M)$ *be two t-PNIDs. Their* composition *is defined by:*

$$N \uplus M = (P_N \cup P_M, T_N \cup T_M, F_N \cup F_M, \alpha_N \cup \alpha_M, \beta_N \cup \beta_M)$$

It is easy to see that the composition of two t-JNs does not automatically result in a t-JN. Consider nets in Fig. 4. It is easy to see that both N and M can be obtained by applying R2 from Definition 10. However, their composition cannot be reduced to a single transition by consecutively applying rules from Definition 10.

A more surprising observation is that composing type projections of a t-JN may not result in a t-JN. Take for example the net from Fig. 5. Both its projections on $\{\lambda_1\}$ and $\{\lambda_2\}$ are t-JNs. However, bringing them together using the composition operator results in a t-PNID that is not t-JN: indeed, since the "copies" of place p appear in three places, and all such copies have same pre- and post-sets (and only differ by their respective types), it is impossible to apply identifier elimination rule *R6* from Definition 10.

As one may observe from the above example, the only difference between $[p_{xy}, \langle \lambda_1, \lambda_2 \rangle]$ and its copies p_x and p_y is in their respective types, whereas the identifiers carried by p_x and p_y are always contained in p_{xy}, and thus both p_x and p_y can be seen as subsidiary with respect to p_{xy}. We formalize this observation using the notion of *minor places*: a place p is minor to some place q if both p and q have identical pre- and post-sets, and the type of q subsumes the one of p.

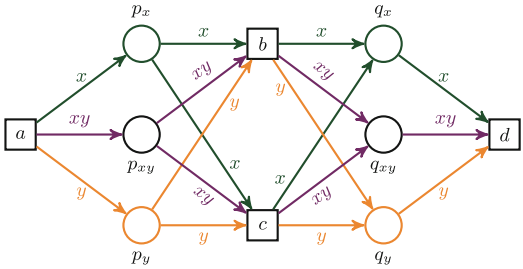


Fig. 5. Composition of the projections on $\{\lambda_1\}$, $\{\lambda_2\}$ and $\{\lambda_1, \lambda_2\}$ on the t-JN $(a; [p, \langle x, y\rangle]; (b||c); [q, \langle x, y\rangle]; d)$. Here, type assignments are as follows: $\alpha(p_x) = \alpha(q_x) = \lambda_1$, $\alpha(p_y) = \alpha(q_y) = \lambda_2$ and $\alpha(p) = \alpha(q) = \lambda_1\lambda_2$.

Definition 13 (Minor places). *Let $N = (P_N, T_N, F_N, \alpha, \beta)$ be a t-PNID. A place $p \in P$ is* minor to *a place $q \in P$ iff the following holds:*

- ${}^\bullet p = {}^\bullet q$, $p^\bullet = q^\bullet$ *and* $\alpha(p) \subset \alpha(q)$;
- $\beta((t,p)) = \beta((t,q))|_{\mathtt{type}^{-1}(\alpha(p))}$, *for each* $t \in {}^\bullet p$;
- $\beta((p,t)) = \beta((q,t))|_{\mathtt{type}^{-1}(\alpha(p))}$, *for each* $t \in p^\bullet$.

◁

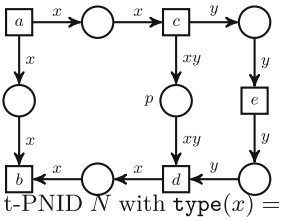
We show next that minor places can be added or removed without altering the overall behavior of the net.

Lemma 2. *Let $N = (P, T, F, \alpha, \beta)$ be a t-PNID with initial marking m_0 s.t. $m_0(p) = m_0(q) = \emptyset$, for $p, q \in P$, where p is minor to q. Let $N' = (P \setminus \{p\}, T, F \setminus (\{(p,t) | t \in p^\bullet\} \cup \{(t,p) | t \in {}^\bullet p\}), \alpha, \beta)$ be a t-PNID obtained by eliminating from N place p. Then $\Gamma_{N,m_0} \sim^r \Gamma_{N',m_0}$.* ◁

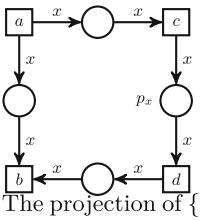
Proof. (sketch) It is enough to define a relation $Q \subseteq \mathcal{R}(N, m_0) \times \mathcal{R}(N', m_0)$ s.t. $(m, m') \in Q$ iff $m(r) = m'(r)$, for $r \in P \setminus \{p\}$, and $m(p)(\mathtt{id}) = m'(q)(\mathtt{id})$, for all $\mathtt{id} \in \mathtt{C}(p)$, and $|m(p)| = |m'(q)|$. Then the lemma statement directly follows from the firing rule of t-PNIDs and that pre- and post-sets of p and q coincide. ■

Let us now address the reconstructability property. In a nutshell, a net is reconstructable if composing all of its type projections returns the same net. This property is not that trivial to obtain. For example, let us consider singleton projections (that is, projections $\pi_{\{\lambda\}}(N)$ obtained for each $\lambda \in \mathtt{type}_\Lambda(N)$) of the net in Fig. 6. It is easy to see that such projections "ignore" interactions between objects (or system components). Thus, the composition of the singleton projections $\pi_{\{\lambda_1\}}(N)$ and $\pi_{\{\lambda_2\}}(N)$ from Fig. 6 does not result in a model that merges p_x and p_y in one place as the composition operator cannot recognize component interactions between such projections. This is reflected in Fig. 6d.

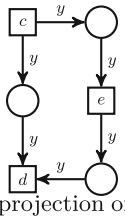
To be able to reconstruct the original model from its projections (or at least do it approximately well), one needs to consider a projection reflecting component interactions. In the case of the net from Fig. 6a, its non-singleton projection



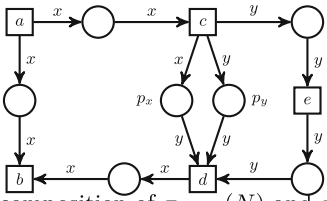
(a) t-PNID N with $\texttt{type}(x) = \lambda_1$ and $\texttt{type}(y) = \lambda_2$



(b) The projection of $\{\lambda_1\}$ on N

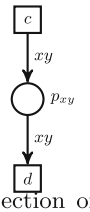


(c) The projection of $\{\lambda_2\}$ on N

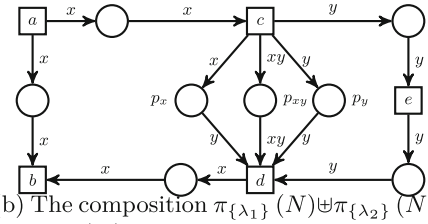


(d) The composition of $\pi_{\{\lambda_1\}}(N)$ and $\pi_{\{\lambda_2\}}(N)$

Fig. 6. t-PNID N (6a), its singleton projections and their composition



(a) The projection of $\{\lambda_1, \lambda_2\}$ on N from Figure 6a



(b) The composition $\pi_{\{\lambda_1\}}(N) \uplus \pi_{\{\lambda_2\}}(N) \uplus \pi_{\{\lambda_1,\lambda_2\}}(N)$ for N from Figure 6a

Fig. 7. Adding the projection $\pi_{\{\lambda_1,\lambda_2\}}(N)$ reflecting interactions to the composition results in the original net N modulo places minor to p (such as p_x and p_y).

$\pi_{\{\lambda_1,\lambda_2\}}(N)$ is depicted in Fig. 7a. Now, using this projection we can obtain a composition (see Fig. 7b) that closely resembles N. Notice that, in this composition, copies of the interaction place p appear three times as places p_x, p_y and p_{xy}, respectively. It is also easy to see that places p_x and p_y are minor to p_{xy}, and $\alpha(p) = \alpha(p_{xy})$ witnesses that $\pi_{\{\lambda_1,\lambda_2\}}(N)$ is the maximal projection defined over types of N s.t. the correct type of p is "reconstructed". This leads us to the following result stipulating the reconstructability property of typed Jackson nets.

Theorem 2. *Let $N = (P, T, F, \alpha, \beta)$ be a t-JN. Then $\Gamma_{N,\emptyset} \sim^r \Gamma_{N',\emptyset}$, where* $N' = \biguplus_{\emptyset \subset \Upsilon \subseteq \texttt{type}_\Lambda(N)} \pi_\Upsilon(N)$. ◁

Proof. (sketch) The proof immediately follows from the next observation. Among all possible projections, for each place $p \in P$ there exists a projection $\pi_\Upsilon(N)$ such that $\alpha(p) = \Upsilon$. This also means that $\pi_\Upsilon(N)$ contains p and that all other

projections $\pi_{\Upsilon'}(N)$ with $\Upsilon' \subset \Upsilon$ will at most include the minors of p. Following Definition 12, it is easy to see that the composition of all the projections yields a t-JN identical to N modulo additional place minors introduced by some of the projections. Showing that the obtained net is bisimilar to N can be done by analogy with Lemma 2. ■

Notice that the above result can be made stronger if all the additional minors (i.e., minors that were not present originally in N) are removed using reduction rules from Definition 10. For simplicity, given a t-PNID N with the set of places P, we denote by $\lfloor P \rfloor$ the set of its minor places.

Corollary 1. *Let N be a t-JN and N' is as in Theorem 2. Then $(N, \emptyset) \leftrightsquigarrow (N', \emptyset)$, if $\lfloor P \rfloor = \lfloor P' \rfloor$, where P and P' are respectively the sets of places of N and N'.* ◁

The above result can be obtained by complementing the proof of Theorem 2 with a step that applies finitely many t-JN reduction rules to all the minor places that are in N' and not in N.

5 A Framework for Rediscoverability

In the previous section, we showed that t-JNs enjoy the reconstructability property: given a t-JN, a composition of *all* its (proper) type projections yields a t-JN that is strongly bisimilar to the original one.[2]

In this section, we propose a framework to rediscover systems of interacting processes that rely on this property. The framework builds upon a divide and conquer strategy [21]. The first step of the approach is to divide the event logs over all possible projections. For this, we translate the notion of event logs to event logs of interacting systems, and show that if these event logs are generated by a t-JN, projections on these event logs have a special property: the projected event log can be replayed by the projected net. In other words, there is no distinction between the projection on the event log, or that the projected net generated the event log. This observation forms the basis of the proposed framework for rediscoverability. In the second step, we conquer the discoverability problem of the system of interacting processes by first discovering a model for each of the projections, and then composing these projections into the original system. If the event log and discovery algorithm guarantee the defined properties, composition yields rediscoverability.

5.1 Event Logs and Execution Traces

In process discovery, an event log is represented as a (multi)set of sequences of events (called traces), where each sequence represents an execution history of a

[2] Such nets are also isomorphic if minor places of the composition are removed by consecutively applying the reduction rules from Definition 10.

Table 1. Firing sequence for the t-PNID in Fig. 1

transition	x	y	z
A	$p1$		
A	$p2$		
T			$c1$
G		$o1$	$c1$
C	$p1$		
E	$p2$	$o1$	

transition	x	y	z
T			$c2$
H		$o1$	
L		$o1$	
J		$o1$	
B	$p2$		
O		$o1$	

transition	x	y	z
D	$p1$		
V			$c2$
K		$o1$	
Z		$o1$	$c1$
V			$c1$
B	$p1$		

process instance. Traditional process discovery assumes the process to be a WF-net. Consequently, each trace in an event log should correspond to a sequence of transition firings of the workflow net. If this is the case, the event log is said to be generated by the WF-net. We generalize this notion to marked Petri nets.

Definition 14 (Event Log). *Given a set of transitions T, a set of traces $L \subseteq T^*$ is called an* event log. *An event log L* is generated by *a marked Petri net (N, m) if $(N, m)[\sigma\rangle$ for all $\sigma \in L$, i.e., $L \subseteq \mathcal{L}(N, m_0)$.* ◁

Each sequence in a single process event log assumes to start from the initial marking of the WF-net. A marked t-PNID, instead, represents a continuously executing system, for which, given a concrete identifier, exists a single observable execution that can be recorder in an event log. Thus, event logs are partial observations of a larger execution within the system: an event log for a certain type captures only the relevant events that contain identifiers of that type, and stores these in order of their execution. Since each transition firing consists of a transition and a binding, a t-PNID firing sequence induces an event log for each set of types Υ. Intuitively, this induced event log is constructed by a filtering process. For each possible identifier vector for Υ we keep a firing sequence. Each transition firing is inspected, and if its binding satisfies an identifier vector of Υ, it is added to the corresponding sequence.

Definition 15 (Induced Event Log). *Let (N, m_0) be a marked t-PNID. Given a non-empty set of types $\Upsilon \subseteq \mathtt{type}_\Lambda(N)$, the* Υ-induced event log *of a firing sequence $\eta \in \mathcal{L}(N, m_0)$ is defined by: $Log_\Upsilon(\eta) = \{\eta_{|i} \mid i \in (Id(\eta) \cap I(\Upsilon))^{|\Upsilon|}\}$, where $\eta_{|i}$ is inductively defined by (1) $\epsilon_{|i} = \epsilon$, (2) $(\langle(t, \psi)\rangle \cdot \eta)_{|i} = \langle(t, \psi)\rangle \cdot \eta_{|i}$ if $supp(i) \subseteq \text{RNG}(\psi)$, and (3) $(\langle(t, \psi)\rangle \cdot \eta)_{|i} = \eta_{|i}$ otherwise.* ◁

Different event logs can be induced from a firing sequence. Consider, for example, the firing sequence of the net from Fig. 1 represented as table in Table 1. As we cannot deduce the types for each of the variables from the firing sequences in Table 1, we assume that there is a bijection between variables and types, i.e., that each variable is uniquely identified by its type, and vice-versa. Like that, we can create an induced log for each variable, as the type and variable name are interchangeable. For example, the x-induced event log is $Log_{\{x\}} = \{\langle A, E, B\rangle, \langle A, C, D, B\rangle\}$, and the z-induced event log is

$Log_{\{z\}} = \{\langle T, G, Z, V\rangle, \langle T, V\rangle\}$. Similarly, event logs can be also induced for combinations of types. In this example, the only non-empty induced event logs on combined types are $Log_{\{y,z\}} = \{\langle G, Z\rangle\}$ and $Log_{\{x,y\}} = \{\langle E\rangle\}$.

As the firing sequence in Table 1 shows, transition firings (and thus also events) only show bindings of variables to identifiers. For example, for firing G with binding $y \mapsto o1$ and $z \mapsto c1$, it is not possible to derive the token types of the consumed and produced tokens directly from the table. Therefore, we make the following assumptions for process discovery on t-PNIDs:

1. There are no "black" tokens: all places carry tokens with at least one type, and all types occur at most once in a place type, i.e., all places refer to at least one process instance.
2. There is a bijection between variables and types, i.e., for each type exactly one variable is used.
3. A Gödel-like number $\mathscr{G}$ is used to order the types in place types, i.e., for any place p, we have $\mathscr{G}(\alpha(p)(i)) < \mathscr{G}(\alpha(p)(j))$ for $1 \leq i < j \leq |\alpha(p)|$ and $p \in P$.

5.2 Rediscoverability of Typed Jackson Nets

Whereas traditional process discovery approaches relate events in an event log to a single object: the process instance, object-centric approaches can relate events to many objects [12]. Most object-centric process discovery algorithms (e.g., [5,17]) use a divide and conquer approach, where "flattening" is the default implementation to divide the event data in smaller event logs. The flattening operation creates a trace for each object in the data set, and combines the traces of objects of the same type in an event log. As we have shown in Sect. 4, singleton projections, i.e., those just considering types in isolation, are insufficient to reconstruct the t-JN that induced the object-centric event log. A similar observation is made for object-centric process discovery (cf. [3,5,7]): flattening the event data into event logs generates inaccurate models. Instead, reconstructability can only be achieved if all possible combinations of types are considered. Hence, for a divide and conquer strategy, the divide step should involve all possible combinations of types, i.e., each interaction between processes requires their own event log. In the remainder of this section, we show that if all combinations of types are considered, flattening is possible, and traditional process discovery algorithms can be used to rediscover a system of interacting processes.

For a system of interacting processes, we consider execution traces, i.e., a firing sequence from the initial marking. Like that, event logs for specific types or combinations of types are induced from the firing sequence. The projection of the system on a type or combinations of types, results again in a t-JN. Similarly, if we project a firing sequence of a t-JN N on a set of types Υ, then this projection is a firing sequence of the Υ-projection on N. The property follows directly from the result that t-JN N is weakly simulated by its Υ-projection.

Lemma 3. *Let N be a t-JN, and let $\Upsilon \subseteq \mathtt{type}_\Lambda(N)$. Then $\hat{\mathtt{H}}_U(\Gamma_{N,\emptyset}) \preccurlyeq^r \Gamma_{\pi_\Upsilon(N),\emptyset}$, with $U = T_N \setminus T_\Upsilon$.* ◁

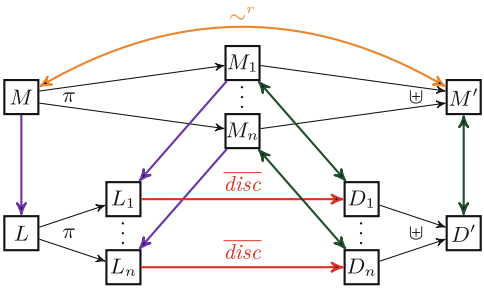


Fig. 8. Framework for rediscoverability of typed Jackson Nets. Model M generates an event log L. Log projections $L_1 \dots L_n$ are generated from projected nets $M_1 \dots M_n$. Discovery algorithm *disc* results in nets $D_1 \dots D_n$, isomorphic to $M_1 \dots M_n$, which can be composed in D'. D' is isomorphic to M' and thus to M.

Proof. (sketch) Let $N_\Upsilon = \Upsilon_{|N} = (P_\Upsilon, T_\Upsilon, F_\Upsilon, \alpha_\Upsilon, \beta_\Upsilon)$. We can define a relation $Q \subseteq \mathbb{M}(N) \times \mathbb{M}(\pi_\Upsilon(N))$ s.t. $Q(m)(p)(a_{|I(\Upsilon)}) = m(p)(a)$ if $p \in P_\Upsilon$ and $Q(m)(p) = m(p)$ otherwise. The rooted weak bisimulation of Q follows directly from the firing rule of t-PNIDs. ■

As the lemma shows, projecting a firing sequence yields a firing sequence for the projected net. A direct consequence of the simulation relation is that, no matter whether we induce an event log from a firing sequence on the original net, or induce it from the projected firing sequence, the resulting event logs are the same.

Corollary 2. *Let (N, m_0) be a marked t-PNID. Given a set of types $\Upsilon \subseteq \mathtt{type}_\Lambda(N)$. Then $Log_\Upsilon(\eta) = Log_\Upsilon(\pi_\Upsilon(\eta))$.* ◁

Hence, it is not possible to observe whether an induced event log stems from the original model, or from its projection. Note that the projection may exhibit more behavior, so the reverse does not hold. In general, not any induced event log from the projection can be induced from the original model.

In general, a projection does not need to be an atomic t-JN (that is, a t-JN that can be reduced by applying rules from Definition 10 to a single transition). However, if the projection is atomic, then its structure is a transition-bordered WF-net: a WF-net that, instead of having source and sink places, has a set of start and finish transitions, such that pre-sets (resp., post-sets) of start (resp., finish) transitions are empty. The closure of a transition-bordered WF-net is constructed by adding a new source place i so that each start transition consumes from i, and a new sink place f so that each finish transition produces in f.

Lemma 4. *Let N be a t-JN and $\pi_\Upsilon(N) = (P_\Upsilon, T_\Upsilon, F_\Upsilon, \alpha_\Upsilon, \beta_\Upsilon)$ for some $\Upsilon \subseteq \mathtt{type}_\Lambda(N)$ such that $\pi_\Upsilon(N)$ is atomic. Let $\eta \in \mathcal{L}(N, \emptyset)$ be a firing sequence. Then $Log_\Upsilon(\eta)$ is generated by $(N_\Upsilon, \emptyset)$ with $N_\Upsilon = (P_\Upsilon \cup \{i, f\}, T_\Upsilon, F_\Upsilon\{(i,t) \mid {}^\bullet t = \emptyset\} \cup \{(t,f) \mid t^\bullet = \emptyset\})$.* ◁

Proof. (sketch) Let $\sigma \in Log_\Upsilon(\eta)$. By construction, each firing sequence in $Log_\Upsilon(\eta)$ has some corresponding identifier vector that generated the sequence. Assume $\vec{v} \in \mathcal{I}^{|\Upsilon|}$ is such a vector for σ.

Observe that for any transition $t \in T$ if ${}^\bullet t = \emptyset$, $Emit(t) \cap \Upsilon \neq \emptyset$, and similarly, if $t^\bullet = \emptyset$, $Collect(t) \cap \Upsilon \neq \emptyset$. As N is identifier sound, only ${}^\bullet\sigma(1) = \emptyset$ and $\sigma(|\sigma|)^\bullet = \emptyset$. Define relation $R = \{(M, m) \mid \forall p \in P : M(p)(\upsilon) = m(p)\}$ and $U = \{(t, \psi) \mid \upsilon \not\subseteq \text{RNG}(\psi)\}$, i.e., U contains all transitions that do not belong to σ. Then R is a weak simulation, i.e., $\hat{\text{H}}_U(\Gamma_{N,\emptyset}) \preccurlyeq^r_R \Gamma_{N_\Upsilon,\emptyset}$ and thus $(N_\Upsilon, \emptyset)[\sigma\rangle$. ■

Given a set of types Υ, if its projection is atomic, the projection can be transformed into a workflow net, and for any firing sequence of the original net, this WF-net can generate the Υ-induced event log. Suppose we have a discovery algorithm *disc* that can rediscover models, i.e., given an event log L that was generated by some model M, then *disc* returns the original model. Rediscoverability of an algorithm requires some property $P_{disc}(M)$ on the generating model M, and some property $Q_{disc}(L, M)$ on the quality of event log L with respect to the generating model M. In other words, $P(M)$ and $Q(L, M)$ are premises to conclude rediscoverability for discovery algorithm *disc*. For example, α-miner [22] requires for $P(M)$ that model M is well-structured, and for $Q(L, M)$ that event log L is directly-follows complete with respect to model M. Similarly, Inductive Miner [16] requires the generating model M to be a process tree without silent actions or self-loops ($P(M)$), and that event log L is directly-follows complete with respect to the original model M ($Q(L, M)$).

Definition 16 (Rediscovery). *An algorithm disc can* rediscover *WF-net* $W = (P, T, F, in, out)$ *from event log* $L \subseteq T^*$ *if* $P_{disc}(W)$ *and* $Q_{disc}(L, W)$ *imply* $disc(L) \leftrightsquigarrow W$. ◁

Thus, suppose there exists a discovery algorithm *disc* that is – under conditions P and Q – able to reconstruct a workflow model given an event log. In other words, given an event log L generated by some model M, *disc* returns a model that is isomorphic to the generating model. Now, suppose we have a firing sequence η of some t-JN N, and some projection Υ. Then, if $P(\pi_\Upsilon(N))$, and $Q(Log_\Upsilon(\eta), \pi_\Upsilon(N))$, then *disc* returns a model that is isomorphic to the closure of $\pi_\Upsilon(N)$, as *disc* only returns WF-nets. With $\overline{disc}$ we denote the model where the source and sink places are removed, i.e., $\overline{disc} \leftrightsquigarrow \pi_\Upsilon(N)$. Then, as shown in Fig. 8, if we discover for every possible combination of types, i.e., the subset-closed set of all type combinations, a model that is isomorphic to the type-projected model, then the composition results in a model that is bisimilar to the original model.

Theorem 3 (Rediscoverability of typed Jackson Nets). *Let* N *be a t-JN, and let* $\eta \in \mathcal{L}(N, \emptyset)$ *without minor places. Let disc be a discovery algorithm with properties* P *and* Q *that satisfy Definition 16. If for all* $\emptyset \subset \Upsilon \subseteq \texttt{type}_\Lambda(N)$ *the* Υ*-projection is atomic and satisfies conditions* $P(\pi_\Upsilon(N))$ *and* $Q(Log_\Upsilon(\eta)), \pi_\Upsilon(N))$, *then* $\Gamma_{N,\emptyset} \leftrightsquigarrow \Gamma_{N',\emptyset}$ *with* $N' = \biguplus_{\emptyset \subset \Upsilon \subseteq \texttt{type}_\Lambda(N)} \overline{disc}(Log_\Upsilon(\eta))$.

Proof. (sketch) Let $\emptyset \subset \Upsilon \subseteq \mathtt{type}_\Lambda(N)$ be a set of types in N. Since $P(\pi_\Upsilon(N))$ and $Q(Log_\Upsilon(\eta)), \pi_\Upsilon(N))$the closure of $\pi_\Upsilon(N)$ and $disc(Log_\Upsilon(\eta))$ are isomorphic. From the closure, places in and out exist with ${}^\bullet in = \emptyset = out^\bullet$. As the nets are isomorphic, we have $\Upsilon_{|N} \leftrightsquigarrow \overline{disc}(Log_\Upsilon(\eta))$. Combining the results gives $\biguplus_{\emptyset \subset \Upsilon \subseteq \mathtt{type}_\Lambda(N)} \overline{disc}(Log_\Upsilon(\eta)) \leftrightsquigarrow \biguplus_{\emptyset \subset \Upsilon \subseteq \mathtt{type}_\Lambda(N)} \pi_\Upsilon(N)$. The statement then follows directly from Corollary 1. ■

6 Conclusion

In this paper, we studied typed Jackson Nets to model systems of interacting processes, a class of well-structured process models describing manipulations of object identifiers. As we show, this class of nets has an important property of reconstructability. In other words, the composition of the projections on all possible type combinations returns the model of the original system. Ignoring the interactions between processes results in less accurate, or even wrong, models. Similar problems occur in the discovery of systems of interacting processes, such as object-centric process discovery, where event logs are flattened for each object.

This paper provides a formal foundation for the composition of block-structured nets, and uses this to develop a framework for the discovery of systems of interacting processes. We link the notion of event logs used for process discovery to system executions, and show that it is not possible to observe whether an event log is generated by a system of interacting processes, or by a projection of the system. These properties form the key ingredients of the framework. We show under what conditions a process discovery algorithm (that guarantees rediscoverability) can be used to discover the individual processes and their interactions, and how these can be combined to rediscover a model of interacting processes that is bisimilar to the original system that generated the event logs.

Although typed Jackson Nets have less expressive power than formalisms like Object-centric Petri nets [5], proclets [11] or interacting artifacts [17], this paper shows the limitations and potential pitfalls of discovering interacting processes. This work aims to lay formal foundations for object-centric process discovery. As a next step, we plan to implement the framework and tune our algorithms to discover useful models from industrial datasets.

Acknowledgements. Artem Polyvyanyy was in part supported by the Australian Research Council project DP220101516.

References

1. Aalst, W.M.P.: Workflow verification: finding control-flow errors using petri-net-based techniques. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45594-9_11
2. Aalst, W.M.P.: Verification of workflow nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63139-9_48

3. Aalst, W.M.P.: Object-centric process mining: dealing with divergence and convergence in event data. In: Ölveczky, P.C., Salaün, G. (eds.) SEFM 2019. LNCS, vol. 11724, pp. 3–25. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30446-1_1
4. van der Aalst, W.M.P.: Foundations of process discovery. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 37–75. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-08848-3_2
5. van der Aalst, W.M.P., Berti, A.: Discovering object-centric petri nets. Fund. Inform. **1–4**(175), 1–40 (2020)
6. van der Aalst, W.M.P., et al.: Soundness of workflow nets: classification, decidability, and analysis. Formal Asp. Comput. **23**(3), 333–363 (2011)
7. Adams, J.N., Park, G., Levich, S., Schuster, D., van der Aalst, W.M.P.: A framework for extracting and encoding features from object-centric event data. In: Troya, J., Medjahed, B., Piattini, M., Yao, L., Fernáindez, P., Ruiz-Cortés, A. (eds.) ICSOC 2022. LNCS, vol. 13740, pp. 36–53. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-20984-0_3
8. Barenholz, D., Montali, M., Polyvyanyy, A., Reijers, H.A., Rivkin, A., van der Werf, J.M.E.M.: On the reconstructability and rediscoverability of typed Jackson nets (extended version) (2023). https://doi.org/10.48550/ARXIV.2303.10039, https://arxiv.org/abs/2303.10039
9. Berti, A., van der Aalst, W.M.P.: OC-PM: analyzing object-centric event logs and process models. Int. J. Softw. Tools Technol. Transf. (2022). https://doi.org/10.1007/s10009-022-00668-w
10. Best, E., Devillers, R., Koutny, M.: The box algebra=petri nets+process expressions. Inf. Comput. **178**(1), 44–100 (2002). https://doi.org/10.1006/inco.2002.3117
11. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: Donatelli, S., Haar, S. (eds.) PETRI NETS 2019. LNCS, vol. 11522, pp. 3–24. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21571-2_1
12. Ghahfarokhi, A.F., Park, G., Berti, A., van der Aalst, W.M.P.: OCEL: a standard for object-centric event logs. In: Bellatreche, L., et al. (eds.) ADBIS 2021. CCIS, vol. 1450, pp. 169–175. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85082-1_16
13. Glabbeek, R.J.: The linear time—branching time spectrum II. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 66–81. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57208-2_6
14. van Hee, K.M., Hidders, J., Houben, G.J., Paredaens, J., Thiran, P.: On the relationship between workflow models and document types. Inf. Syst. **34**(1), 178–208 (2009). https://doi.org/10.1016/j.is.2008.06.003
15. Kopp, O., Martin, D., Wutke, D., Leyman, F.: The difference between graph-based and block-structured business process modelling languages. EMISAJ **4**(1), 3–13 (2009)
16. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_17
17. Lu, X., Nagelkerke, M., van de Wiel, D., Fahland, D.: Discovering interacting artifacts from ERP systems. IEEE Trans. Serv. Comput. **8**(6), 861–873 (2015)
18. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989). https://doi.org/10.1109/5.24143

19. Polyvyanyy, A., van der Werf, J.M.E.M., Overbeek, S., Brouwers, R.: Information systems modeling: language, verification, and tool support. In: Giorgini, P., Weber, B. (eds.) CAiSE 2019. LNCS, vol. 11483, pp. 194–212. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21290-2_13
20. Tour, A., Polyvyanyy, A., Kalenkova, A.A.: Agent system mining: vision, benefits, and challenges. IEEE Access **9**, 99480–99494 (2021)
21. Tour, A., Polyvyanyy, A., Kalenkova, A.A., Senderovich, A.: Agent miner: an algorithm for discovering agent systems from event data. CoRR abs/2212.01454 (2022)
22. van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
23. van der Werf, J.M.E.M., Rivkin, A., Polyvyanyy, A., Montali, M.: Data and process resonance. In: Bernardinello, L., Petrucci, L. (eds.) PETRI NETS 2022. LNCS, vol. 13288, pp. 369–392. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-06653-5_19