Encoding Petri Nets into CCS

Benjamin Bogø[1] (✉) [0009−0000−2192−3291], Andrea Burattin[1] [0000−0002−0837−0183], and Alceste Scalas[1] [0000−0002−1153−6164]

Technical University of Denmark, Denmark
`{bbog,andbur,alcsc}@dtu.dk`

Abstract. This paper explores the problem of determining which classes of Petri nets can be encoded into behaviourally-equivalent CCS processes. Most of the existing related literature focuses on the inverse problem (i.e., encoding process calculi belonging to the CCS family into Petri nets), or extends CCS with Petri net-like multi-synchronisation (Multi-CCS). In this work, our main focus are *free-choice* and *workflow* nets (which are widely used in process mining to describe system interactions) and our target is *plain* CCS. We present several novel encodings, including one from free-choice workflow nets (produced by process mining algorithms like the α-miner) into CCS processes, and we prove that our encodings produce CCS processes that are weakly bisimilar to the original net. Besides contributing new expressiveness results, our encodings open a door towards bringing analysis and verification techniques from the realm of process calculi into the realm of process mining.

Keywords: Petri nets · CCS · Encoding · Bisimulation · Free-choice workflow nets.

1 Introduction

Process calculi and Petri nets are among the most successful tools for the modelling and verification of concurrent systems. The two models have significantly different designs: Petri nets have a more *semantic* flavour, whereas process calculi have a more *syntactic* flavour. This has resulted in significantly different approaches and application fields. In particular, Petri nets have found considerable success in the area of *Workflow Management*, as the theoretical foundation for several *Business Process Management* languages, and in *process mining*, whereas the syntactic nature of process calculi has fostered a rich literature on the static verification of behavioural properties (e.g. via type checking or the axiomatisation of bisimulation relations), often connected to programming languages.

This different focus on semantics-vs.-syntax has naturally encouraged the study of Petri nets as a possible semantic model for process calculi, through the development of various encodings and results of the form: *Petri nets (of the class X) are at least as expressive as the encoded calculus Y*. (For more details, see Section 2.) In this paper we investigate the opposite problem: *Which flavour of Petri nets can be encoded in Milner's Calculus of Communicating Systems*

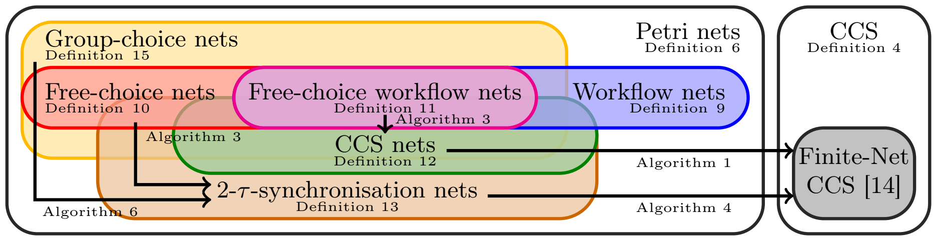


Fig. 1. Overview of the relation between Petri net classes and CCS considered in this paper. The arrows show algorithms for converting one class into another class.

(CCS)? A reason for this investigation is the observation that applications of Petri nets in process mining (e.g. via the α-miner algorithm [3]) often result in rather structured nets (in particular, *free-choice workflow nets* [10,3]) which are reminiscent of what is expressible in CCS. Therefore, we aim at proving whether this intuition is correct. Moreover, besides producing novel expressiveness results, developing an encoding from (selected classes of) Petri nets into CCS could also open new doors towards directly using process calculi in process mining, or applying analysis and verification techniques and tools originally developed for process calculi (e.g. model checkers) to the realm of process mining.

Contributions and structure. Fig. 1 gives an overview of the relation and conversion between Petri nets classes and CCS considered in this paper. We start by presenting related work in Section 2 and preliminaries in Section 3. Then, we present an encoding of *free-choice workflow nets* into weakly bismilar CCS processes (Theorem 4) in Section 4. Here, we also introduce a new class of Petri nets called *group-choice nets* (which include free-choice nets) and show how to encode them into weakly bisimilar CCS processes (Theorem 7). We conclude and outline future work in Section 5. A software tool [7] has been created based on the results of this paper: a web application to import/draw Petri nets, classify them (according to Fig. 1) and encode them into CCS. Due to space limits, proofs and additional materials are available in a separate technical report [8].

2 Related Work

Many process mining algorithms (like the α-miner [3]) take a log of traces of *visible* actions and turn it into a *workflow net* [2]. Workflow nets are Petri nets used to describe how systems interact during a process. These kinds of interactions can also be described by process calculi like CCS with labeled semantics to capture the visible actions. There has been some debate about whether graphs like Petri nets or process calculi like the π-calculus are best for process mining [1]. Most of the existing work builds on Petri nets [11], but there is also work on how to represent patterns in process calculi [16].

Most existing work about encodings between process calculi and Petri nets focus on encoding the former into the latter: e.g., there are encodings from variants of CCS [12,5,4], CSP [6], and finite-control π-calculus [15] to various classes of Petri nets. [15] also briefly describes an encoding from *unlabeled* safe (1-bounded) Petri nets into CCS with reduction semantics; the result of the encoding is claimed weakly bisimilar to the original net. However, applications in process mining require *labelled* semantics.

To our knowledge, encodings of Petri nets into process calculi are less explored. Gorrieri and Versari [13] present an extension of CCS called *Multi-CCS*, with the purpose of having a one-to-one correspondence between unsafe P/T Petri nets and Multi-CCS; crucially, Multi-CCS can synchronize multiple processes at a time (like Petri nets) whereas CCS is limited to two synchronizing processes at a time. [13] also presents an encoding from Petri nets into strongly bisimilar Multi-CCS processes; they also show that encoding a restricted class of Petri nets (called *CCS nets*) yields strongly bisimilar *plain* CCS process. In this paper we start from this last result, and explore encodability beyond CCS nets — targeting *plain* CCS only (to enable reusing its well-established techniques e.g. for model checking and axiomatic reasoning), and with an eye torward classes of Petri nets relevant for process mining.

3 Preliminaries: LTSs, Bisimulations, CCS, Petri Nets

This section contains the basic standard definitions used in the rest of the paper.

LTSs and bisimulations. We adopt standard definitions of strong and weak bisimulation between LTS states (Definition 1, 2, and 3, based on [14]).

Definition 1 (Labeled transition system). *A* labeled transition system (LTS) *is a triple $(Q, A, \dot{\rightarrow})$ where Q is a set of* states*, A is a set of* actions*, and $\dot{\rightarrow} \subseteq Q \times A \times Q$ is a* labelled transition relation*. The set A may contain a distinguished* internal action *τ, and we dub any other action as* visible*. We write:*

- $q \xrightarrow{\mu} q'$ *iff* $(q, \mu, q') \in \dot{\rightarrow}$
- $q \xrightarrow{a} q'$ *iff* $a \neq \tau$ *and* $(q, a, q') \in \dot{\rightarrow}$ *(note that the action a is not silent)*
- $q \overset{\epsilon}{\Rightarrow} q'$ *iff* $q = q'$ *or* $q \xrightarrow{\tau} \cdots \xrightarrow{\tau} q'$ *(i.e., q can reach q' in 0 or more τ-steps)*
- $q \overset{a}{\Rightarrow} q'$ *iff* $q \overset{\epsilon}{\Rightarrow}\xrightarrow{a}\overset{\epsilon}{\Rightarrow} q'$ *(q can reach q' via one a-step + 0 or more τ-steps)*

We say that q is a deadlock *if there are no transitions from q. A* divergent path *is an infinite sequence of LTS states $q_1, q_2, \ldots$ such that $q_i \xrightarrow{\tau} q_{i+1}$.*

Definition 2 (Strong bisimulation). *A* strong bisimulation *between two LTSs $(Q_1, A_1, \dot{\rightarrow}_1)$ and $(Q_2, A_2, \dot{\rightarrow}_2)$ is a relation $\mathcal{R} \subseteq Q_1 \times Q_2$ where, if $(q_1, q_2) \in \mathcal{R}$:*

$$\forall q_1' : q_1 \xrightarrow{\mu}_1 q_1' \text{ implies } \exists q_2' : q_2 \xrightarrow{\mu}_2 q_2' \text{ and } (q_1', q_2') \in \mathcal{R}$$
$$\forall q_2' : q_2 \xrightarrow{\mu}_2 q_2' \text{ implies } \exists q_1' : q_1 \xrightarrow{\mu}_1 q_1' \text{ and } (q_1', q_2') \in \mathcal{R}$$

We say that q and q' are strongly bisimilar *or simply* bisimilar *(written $q \sim q'$) if there exists a bisimulation $\mathcal{R}$ with $(q, q') \in \mathcal{R}$.*

Definition 3 (Weak bisimulation). *A* weak bisimulation *between two LTSs* $(Q_1, A_1, \dot{\rightarrow}_1)$ *and* $(Q_2, A_2, \dot{\rightarrow}_2)$ *is a relation* $\mathcal{R} \subseteq Q_1 \times Q_2$ *where, if* $(q_1, q_2) \in \mathcal{R}$*:*

$$\forall q_1' : q_1 \xrightarrow{a}_1 q_1' \ \textit{implies} \ \exists q_2' : q_2 \overset{a}{\Rightarrow}_2 q_2' \ \textit{and} \ (q_1', q_2') \in \mathcal{R}$$
$$\forall q_1' : q_1 \xrightarrow{\tau}_1 q_1' \ \textit{implies} \ \exists q_2' : q_2 \overset{\epsilon}{\Rightarrow}_2 q_2' \ \textit{and} \ (q_1', q_2') \in \mathcal{R}$$
$$\forall q_2' : q_2 \xrightarrow{a}_2 q_2' \ \textit{implies} \ \exists q_1' : q_1 \overset{a}{\Rightarrow}_1 q_1' \ \textit{and} \ (q_1', q_2') \in \mathcal{R}$$
$$\forall q_2' : q_2 \xrightarrow{\tau}_2 q_2' \ \textit{implies} \ \exists q_1' : q_1 \overset{\epsilon}{\Rightarrow}_1 q_1' \ \textit{and} \ (q_1', q_2') \in \mathcal{R}$$

We say that q *and* q' *are* weakly bisimilar *(written* $q \approx q'$*) if there is a weak bisimulation* $\mathcal{R}$ *with* $(q, q') \in \mathcal{R}$*.*

CCS. We adopt a standard version of CCS with LTS semantics, including restrictions and defining equations (Definition 4 and 5, based on [14]).

Definition 4 (CCS syntax). *The syntax of CCS is:*

$$\mu ::= \tau \mid a \mid \overline{a} \qquad P ::= \mathbf{0} \mid \mu.Q \mid P + P' \qquad Q ::= P \mid Q \,|\, Q' \mid (\nu a)Q \mid X$$

By Definition 4, an *action* μ can be the silent action τ, a visible action a, or its *co-action* $\overline{a}$. A *sequential CCS process* P can do nothing ($\mathbf{0}$), perform an *action prefix* μ followed by Q ($\mu.Q$), or perform a *choice* ($P + P'$). A *process* Q can be a sequential process P, a *parallel composition* of two processes ($Q \mid Q'$), a *restriction* of action a to scope Q ($(\nu a)Q$), or a *process name* X.

The LTS semantics of CCS is formalised in Definition 5 below, where it is assumed that there is a partial map of *defining equations* $\mathcal{D}$ from process names to processes, i.e., $\mathcal{D}(X) = Q$ means that $\mathcal{D}$ defines the name X as process Q.

Definition 5 (LTS semantics of CCS). *The LTS of a CCS process* Q *with defining equations* $\mathcal{D}$*, written* $LTS(Q, \mathcal{D})$*, has the least transition relation* $\dot{\rightarrow}$ *induced by the rules below:*

$$\textsc{Pref}\frac{}{\mu.Q \xrightarrow{\mu} Q} \qquad \textsc{Sum1}\frac{Q_1 \xrightarrow{\mu} Q_1'}{Q_1 + Q_2 \xrightarrow{\mu} Q_1'} \qquad \textsc{Par1}\frac{Q_1 \xrightarrow{\mu} Q_1'}{Q_1 \mid Q_2 \xrightarrow{\mu} Q_1' \mid Q_2}$$

$$\textsc{Cons}\frac{Q \xrightarrow{\mu} Q'}{X \xrightarrow{\mu} Q'}\ \mathcal{D}(X) = Q \qquad \textsc{Sum2}\frac{Q_2 \xrightarrow{\mu} Q_2'}{Q_1 + Q_2 \xrightarrow{\mu} Q_2'} \qquad \textsc{Par2}\frac{Q_2 \xrightarrow{\mu} Q_2'}{Q_1 \mid Q_2 \xrightarrow{\mu} Q_1 \mid Q_2'}$$

$$\textsc{Com}\frac{Q_1 \xrightarrow{a} Q_1' \quad Q_2 \xrightarrow{\overline{a}} Q_2'}{Q_1 \mid Q_2 \xrightarrow{\tau} Q_1' \mid Q_2'} \qquad \textsc{Res}\frac{Q \xrightarrow{\mu} Q'}{(\nu a)Q \xrightarrow{\mu} (\nu a)Q'}\ \mu \neq a, \overline{a}$$

By rule Pref in Definition 5, actions (a), co-actions ($\overline{a}$), and internal actions (τ) can be executed by consuming the prefix of a sequential process: for example, we have $a.\mathbf{0} \xrightarrow{a} \mathbf{0}$. The rules Sum1 and Sum2 allow for executing either the left or right branch of a choice: for example, we have $Q \xleftarrow{a} a.Q + b.Q' \xrightarrow{b} Q'$. By rule Res, actions and co-actions cannot be executed when restricted; for example, we have $(\nu b)(a.\mathbf{0}) \xrightarrow{a} (\nu b)\mathbf{0}$, whereas b cannot be executed in $(\nu b)(b.\mathbf{0})$. By rule Com, an action can *synchronize* with its co-action, producing an internal τ-action; for example, b can synchronize with $\overline{b}$, so we have $b.\mathbf{0} \mid \overline{b}.\mathbf{0} \xrightarrow{\tau} \mathbf{0} \mid \mathbf{0}$; this also works under restriction (by rule Res), so we have $(\nu b)(b.\mathbf{0} \mid \overline{b}.\mathbf{0}) \xrightarrow{\tau} (\nu b)(\mathbf{0} \mid \mathbf{0})$.

Petri nets. We adopt standard definitions of labelled Petri nets (that we simply call *Petri nets*), marking, and firing rules (Definition 6, 7, and 8, based on [3]). We also highlight two classes of Petri nets commonly used in process mining literature: *workflow nets* (Definition 9) and *free-choice nets* (Definition 10).

Definition 6 (Labelled Petri net). *A* labelled Petri net *is a tuple* (P, T, F, A, σ) *where* P *is a finite set of* places*,* T *is a finite set of* transitions *such that* $P \cap T = \emptyset$*, and* $F \subseteq (P \times T) \cup (T \times P)$ *is a set of directed* edges *from places to transitions or* vice versa*; moreover,* A *is a set of* actions *and* $\sigma : T \to A$ *assigns an action to each transition.*

Notably, Definition 6 only allows for at most one (unweighted) edge between each pair of places and transitions, and does *not* allow co-actions in A: we keep co-actions exclusive to CCS (Definition 4) to avoid renaming in our encodings.

Definition 7 (Marking). *A* marking *of a Petri net* (P, T, F, A, σ) *is a mapping* $M : P \to \mathbb{N}$ *from each place* $p \in P$ *to the number of* tokens *in* p *(may be 0).*

Definition 8 (Firing rule). *Given a Petri net* (P, T, F, A, σ) *and a marking* $M : P \to \mathbb{N}$*, a transition* $t \in T$ *is* enabled *if all places with an edge to* t *have tokens in* M*. Transition* t *can* fire *if enabled, and this firing consumes one token from all places with an edge to* t*, emits a label* $\sigma(t)$*, and produces one token for all places with an edge from* t*. This results in an updated marking* M'*.*

A Petri net $N = (P, T, F, A, \sigma)$ and an initial marking M_0 yields $LTS(N, M_0) = (Q, A, \dot{\to})$ where the states (Q) and the transition relation $(\dot{\to})$ are derived using the firing rule (Definition 8) until all enabled transitions t in all reachable markings are added to $\dot{\to}$, with $\sigma(t)$ as transition label.

Process mining algorithms like the α-miner typically produce *workflow nets* (Definition 9) that are used to describe end-to-end processes with a clear start and completion. In practical applications, such workflow nets are often *free-choice*[1] (Definition 10 and 11) where all choices are made by a single place — meaning that transitions can at most fight for one token in order to fire. Fig. 2–5 show examples of Petri nets with small differences resulting in different classes.

Definition 9 (Workflow net [3, Definition 2.8]). *A Petri net* (P, T, F, A, σ) *is a* workflow net *iff it satisfies the following three properties:*

- ***Object creation****:* P *has an* input place i *with no ingoing edges.*
- ***Object completion****:* P *has an* output place o *with no outgoing edges.*
- ***Connectedness****: For every* $v \in P \cup T$*, there exists a directed path of edges from* i *to* o *that goes through* v*.*

Definition 10 (Free-choice net [10]). *A Petri net* (P, T, F, A, σ) *is a* free-choice net *iff it satisfies the following two properties:*

[1] The α-miner actually returns a further subclass of free-choice workflow nets called *structured workflow nets* [3].

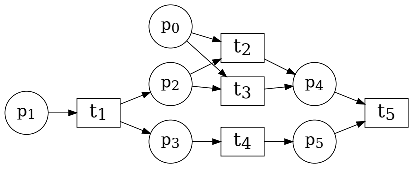


Fig. 2. Neither free-choice or workflow net.

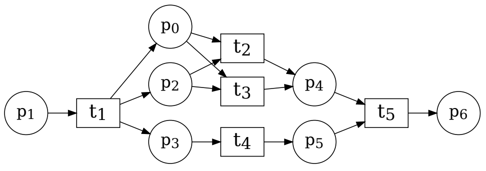


Fig. 3. Workflow but not free-choice net.

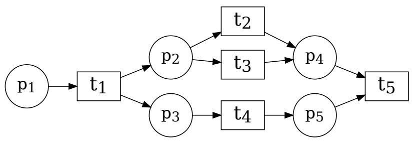


Fig. 4. Free-choice but not workflow net.

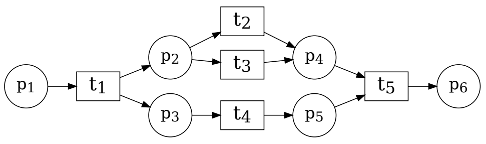


Fig. 5. Free-choice net and workflow net.

– ***Unique choice****: All places $p \in P$ with more than one outgoing edge only have edges to transitions with exactly one ingoing edge (edge from p).*
– ***Unique synchronisation****: All transitions $t \in T$ with more than one ingoing edge only have edges from places with exactly one outgoing edge (edge to t).*

Definition 11 (Free-choice workflow net). *If a Petri net is both a workflow net and free-choice, we call it a* free-choice workflow net*.*

The *connectedness* property in Definition 9 implies that all transitions in free-choice workflow nets have at least one ingoing edge and at least one outgoing edge. The same applies to all places except the special places i and o that respectively has no ingoing edges (i) and no outgoing edges (o). The *unique choice and synchronisation* properties in Definition 10 ensure that every choice is separated from all synchronisations and *vice versa.* This does *not* rule out cycles but *unique choice* restricts all outgoing edges *leaving* a cycle to lead to transitions with exactly one ingoing edge (because there is always one edge *continuing* the cycle). For instance, adding a new transition t_7 in Fig. 5 to form the cycle $(p_3, t_4, p_5, t_7, p_3)$ would violate *unique choice* (and *unique synchronisation*) because the edge (p_5, t_5) leaves the cycle but t_5 has two ingoing edges. Adding the cycle (p_3, t_7, p_3) is allowed because t_4 only has one ingoing edge.

4 Encoding Petri Nets into CCS, Step-by-Step

This section introduces our main contribution: an encoding into CCS of a superclass of free-choice nets, that we call *group-choice nets* (Definition 15); we prove that our encoding is correct, i.e., a Petri net and its encoding are weakly bisimilar and without added divergent states (Theorem 7). To illustrate the encodings and result, we proceed through a series of steps: a series of encoding algorithms into CCS for progressively larger classes of Petri nets. (See Fig. 1 for an outline.)

We begin (in Section 4.1) with the class of *CCS nets* (Definition 12), and Algorithm 1 that encodes such nets into strongly bisimilar CCS processes (Theorem 1).[2] Then (in Section 4.2) we develop a novel transformation from *free-choice workflow nets* (Definition 11) to weakly bismilar CCS nets using Algorithm 3 (Theorem 3). The composition of Algorithm 3 and Algorithm 1 then encodes free-choice workflow nets into weakly bismilar CCS processes (Theorem 4). In Section 4.3, we generalise CCS nets into *2-τ-synchronisation nets* (Definition 13, allowing for transitions with no ingoing edges) and we present Algorithm 4 to encode such nets into strongly bisimilar CCS processes (Theorem 6). Finally (cf. Section 4.4), we generalize free-choice nets to a new class called *group-choice nets* (Definition 15) and present Algorithm 6 that, composed with Algorithm 4, encodes group-choice nets into weakly bismilar CCS processes (Theorem 7).

4.1 Encoding CCS Nets into CCS Processes

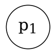
A challenge in encoding Petri nets into CCS processes is that a transition in a Petri net can consume tokens from any number of places in a single step — whereas the semantics of CCS (Definition 5) only allow for executing a single action or a synchronisation between two processes in each step. In other words, Petri nets are able to perform n-ary synchronisation, while CCS can only perform 2-ary synchronisation. Therefore, as a stepping stone towards our main result, we adopt *CCS nets* from [13], whose synchronisation capabilities match CCS.

Definition 12 (CCS net [13]). *A Petri net (P, T, F, A, σ) is a* CCS net *iff each transition $t \in T$ has one or two ingoing edges — and in the latter case, $\sigma(t) = \tau$.*

The key insight behind Definition 12 is that transitions with two ingoing edges must be labeled with τ to have a 1-to-1 correspondence to synchronisation in CCS (that also emits a τ). Fig. 12 shows a CCS net that later will be encoded.

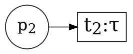
Algorithm 1 encodes a CCS net (P, T, F, A, σ) and marking M_0 into a CCS process Q and its defining equations $\mathcal{D}$, where $\mathcal{D}$ defines a process named X_p for each place $p \in P$. The idea is that each token at place p is encoded as a parallel replica of the process X_p. We illustrate the algorithm in the next paragraphs.

[2] The results in Section 4.1 can be also derived from [13], but here we provide a direct encoding algorithm, statements, and proofs for plain CCS, without using Multi-CCS.



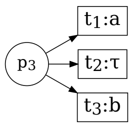
$X_{p_1} := \mathbf{0}$

Fig. 6. Place with 0 outgoing edges and its encoding.



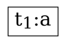
$X_{p_2} := Y_{t_2}$

Fig. 7. Place with 1 outgoing edge and its encoding.

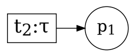


$X_{p_3} := Y_{t_1} + Y_{t_2} + Y_{t_3}$

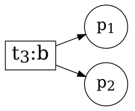
Fig. 8. Place with 3 outgoing edges and its encoding.



$Y_{t_1} := a.\mathbf{0}$



$Y_{t_2} := \tau.X_{p_1}$



$Y_{t_3} := b.(X_{p_1} \mid X_{p_2})$

Fig. 9. Encoding of transition with 0 outgoing edges.

Fig. 10. Encoding of transition with 1 outgoing edge.

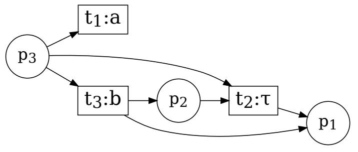
Fig. 11. Encoding of transition with 2 outgoing edges.

On line 2–4, each place $p \in P$ is encoded as a *choice process* named X_p in $\mathcal{D}$: The choice is among placeholder processes named Y_t, for each transition t with an edge from p. (Notice that the placeholders Y_t are not in the domain of $\mathcal{D}$, but are substituted with sequential CCS processes in the next steps of the algorithm.) The choice process X_p models a token at p that *chooses* which transition it is used for. Fig. 6–8 show examples.

On line 5–7, each transition $t \in T$ with one ingoing edge from a place p^* is encoded as a process with an action prefix (obtained via the labelling function $\sigma(t)$) followed by the *parallel composition* of all processes named X_{p_i}, where place p_i has an ingoing edge from t. Fig. 9–11 show examples. The resulting process $\sigma(t).(X_{p_1} \mid \ldots \mid X_{p_k})$ (line 6) models the execution of the action $\sigma(t)$ followed by the production of tokens for places $p_1, \ldots, p_k$, and such a process is used to substitute the placeholder Y_t in $\mathcal{D}(X_{p^*})$.

Algorithm 1: Encoding from CCS net to CCS process

```
Input  : CCS net $(P, T, F, A, \sigma)$ and marking $M_0 : P \to \mathbb{N}$
Output: CCS process $Q$ and partial mapping of defining equations $\mathcal{D}$
1  $\mathcal{D} \leftarrow$ Empty mapping of defining equations
2  for $p \in P$ do
3      $\mathcal{D}(X_p) \leftarrow (Y_{t_1} + Y_{t_2} + \cdots + Y_{t_k})$ where $\{t_1, t_2, \ldots, t_k\} = \{t \mid (p, t) \in F\}$
4  end
5  for $t \in \{t \mid t \in T$ and $t$ has 1 ingoing edge$\}$ do
6      substitute $Y_t$ in $\mathcal{D}(X_{p^*})$ with $\sigma(t).(X_{p_1} \mid X_{p_2} \mid \ldots \mid X_{p_k})$ where
         $(p^*, t) \in F$ and $\{p_1, p_2, \ldots, p_k\} = \{p \mid (t, p) \in F\}$
7  end
8  $A' \leftarrow \emptyset$
9  for $t \in \{t \mid t \in T$ and $t$ has 2 ingoing edges$\}$ do
10     $A' \leftarrow A' \cup \{s_t\}$ where $s_t$ is a fresh action
11     substitute $Y_t$ in $\mathcal{D}(X_{p^*})$ with $s_t.(X_{p_1} \mid X_{p_2} \mid \ldots \mid X_{p_k})$
12     substitute $Y_t$ in $\mathcal{D}(X_{p^{**}})$ with $\overline{s_t}.\mathbf{0}$ where
         $(p^*, t), (p^{**}, t) \in F$ and $p^* \neq p^{**}$ and $\{p_1, p_2, \ldots, p_k\} = \{p \mid (t, p) \in F\}$
13 end
14 $\{s_1, s_2, \ldots, s_n\} \leftarrow A'$
15 $Q \leftarrow (\nu s_1)(\nu s_2)\ldots(\nu s_n)\Big(X_{p_1}^{M_0(p_1)} \mid X_{p_2}^{M_0(p_2)} \mid \ldots \mid X_{p_{|P|}}^{M_0(p_{|P|})}\Big)$
16 return $(Q,\ \mathcal{D})$
```



$M_0(p_1) = 1,\ M_0(p_2) = 0,\ M_0(p_3) = 2$

Fig. 12. CCS net N and initial marking M_0 with three tokens.

$$
\begin{aligned}
Q &\coloneqq (\nu s_{t_2})(X_{p_1} \mid X_{p_3} \mid X_{p_3})\\
\mathcal{D}(X_{p_1}) &= \mathbf{0}\\
\mathcal{D}(X_{p_2}) &= \overline{s_{t_2}}.\mathbf{0}\\
\mathcal{D}(X_{p_3}) &= s_{t_2}.X_{p_1} + a.\mathbf{0} + b.(X_{p_1} \mid X_{p_2})
\end{aligned}
$$

Fig. 13. Encoding of N and M_0 in Fig. 12 into Q and $\mathcal{D}$ produced by Algorithm 1.

On line 8–13, each transition $t \in T$ with two ingoing edges from p^* and p^{**} (where t has label $\sigma(t) = \tau$, by Definition 12) is encoded as a synchronisation between a fresh action s_t (which prefixes $(X_{p_1} \mid \ldots \mid X_{p_k})$ on line 11 to model production of tokens), and its co-action $\overline{s_t}$ (line 12). The two resulting processes are respectively used to substitute the placeholder Y_t in $\mathcal{D}(X_{p^*})$ and $\mathcal{D}(X_{p^{**}})$.

Finally, on line 15, the initial marking M_0 is encoded into the result CCS process Q, which is the *parallel composition* of one place process X_{p_i} per token at place p_i, under a restriction of each fresh action s_i produced on line 10. Note that we write Q^n for the parallel composition of n replicas of Q, so $Q^3 = Q|Q|Q$, and $Q^1 = Q$, and $Q^0 = \mathbf{0}$. The algorithm also returns the partial mapping $\mathcal{D}$ with the definition of each process name X_p (for each $p \in P$) occurring in Q.

Fig. 13 shows the encoding of the CCS net in Fig. 12. Observe that in X_{p_3}, the transitions t_1 and t_3 (which have one ingoing edge) yield respectively the sub-processes $a.\mathbf{0}$ and $b.(X_{p_1} \mid X_{p_2})$ (produced by line 5–7 in Algorithm 1); instead, t_2 (which has two ingoing edges and label τ) yields both sub-processes $s_{t_2}.X_{p_1}$ in X_{p_3} and $\overline{s_{t_2}}.\mathbf{0}$ in X_{p_2} (produced by line 9–13 in Algorithm 1).

Theorem 1 (Correctness of Algorithm 1). *Given a CCS net $N = (P, T, F, A, \sigma)$ and an initial marking M_0, let the result of applying Algorithm 1 to N and M_0 be the CCS process Q and defining equations $\mathcal{D}$. Then, $LTS(N, M_0) \sim LTS(Q, \mathcal{D})$. The translation time and the size of $(Q, \mathcal{D})$ are bound by $O(|N| + \sum_{p \in P} M_0(p))$.*

Proof. (Sketch, detailed proof in [8].) We define the following relation between markings and processes, and we prove it is a bisimulation:

$$\mathcal{R} \coloneqq \Big\{ \Big(M,\ (\nu s_1)\ldots(\nu s_n)\Big(X_{p_1}^{M(p_1)} \mid \ldots \mid X_{p_{|P|}}^{M(p_{|P|})} \Big) \Big) \ \Big|\ M \in LTS(N, M_0) \Big\}$$

where M is a reachable marking from the initial marking M_0 in N (i.e. M is a state in $LTS(N, M_0)$), and $s_1, \ldots, s_n$ are the restricted names in Q. The intuition is as that for every place p containing $M(p)$ tokens, the CCS process contains $M(p)$ replicas of the process named X_p (written $X_p^{M(p)}$). □

The strong bisimulation result in Theorem 1 ensures that there are no observable differences between the original Petri net and its encoding, so Algorithm 1 does not introduce new deadlocks nor divergent paths. The encoding is also linear in the size of the CCS net N and the size of the initial marking M_0.

4.2 Encoding Free-Choice Workflow Nets into CCS Processes

We now build upon the result in Section 4.1 to prove that *free-choice workflow nets* (Definition 11) are encodable into weakly bisimilar CCS processes. Specifically, we present a stepwise transformation procedure (Algorithm 2 and 3) from a free-choice workflow net into weakly bisimilar CCS net (Theorem 3), and then apply Algorithm 1 to get a weakly bisimilar CCS process (Theorem 4).

It should be noted (as illustrated in Fig. 1) that free-choice (workflow) nets does *not* contain the class of CCS nets: Fig. 12 is not a free-choice net since p_3 has multiple outgoing edges of which one leads to a transition with multiple ingoing edges. However, as will be shown in this section, a free-choice workflow net can be transformed into a CCS net by (non-deterministically) making a binary synchronisation order for each n-ary synchronisation.

An application of Algorithm 2 transforms an input Petri net by reducing the number of ingoing edges of the selected transition t^* (which must have two or more ingoing edges): it selects two distinct ingoing edges (line 1), creates a new τ-transition t^+ with an edge to a new place p^+, and connects the new place p^+ to the selected transition t^* (line 2–5). The cost of the transformation is that a new τ-transition t^+ with two ingoing edges is created. Fig. 14 and 15 show an example application on a free-choice net, where t_1 is selected for transformation with the edges (p_2, t_1) and (p_3, t_1). Algorithm 2 can be iterated until all the original transitions only have one ingoing edge left; transitions with no ingoing edges are left untouched. Hence, transformed free-choice workflow nets will have no transitions with zero ingoing edges, hence Algorithm 1 could be applied.

Algorithm 2: Petri net transition preset reduction

```
Input  : Petri net $(P, T, F, A, \sigma)$, marking $M_0 : P \to \mathbb{N}$ and chosen transition
         $t^* \in T$ with at least two ingoing edges
Output: Petri net $(P', T', F', A', \sigma')$ and marking $M'_0 : P' \to \mathbb{N}$
1 select $(p^*, t^*), (p^{**}, t^*) \in F$ where $p^* \neq p^{**}$
2 $P' \leftarrow P \cup \{p^+\}$ where $p^+ \notin P \cup T$
3 $T' \leftarrow T \cup \{t^+\}$ where $t^+ \notin P' \cup T$
4 $F' \leftarrow (F \setminus \{(p^*, t^*), (p^{**}, t^*)\}) \cup \{(p^*, t^+), (p^{**}, t^+), (t^+, p^+), (p^+, t^*)\}$
5 $N' \leftarrow (P', T', F', A \cup \{\tau\}, \sigma[t^+ \mapsto \tau])$
6 return $(N', M_0[p^+ \mapsto 0])$
```

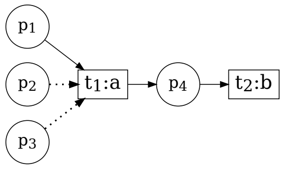


Fig. 14. Free-choice net where t_1 and the dotted edges (p_2, t_1) and (p_3, t_1) are chosen for transformation.

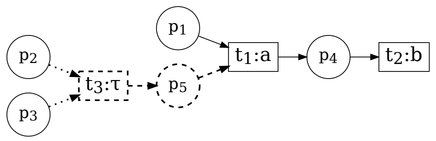


Fig. 15. Free-choice net obtained from Fig. 14 where the dotted edges have been moved and the dashed elements t_3 and p_5 have been added.

Fig. 14 and 15 show that Algorithm 2 does *not* produce a Petri net that is strongly bisimilar to its input: if we only have one token in both p_2 and p_3, then Fig. 15 can fire the τ-transition while Fig. 14 is in a deadlock. However, Theorem 2 below proves that, when applied to free-choice nets (and thereby also free-choice workflow nets), Algorithm 2 produces a weakly bisimilar Petri net. Moreover, by Lemma 1, the returned net remains a free-choice net.

Theorem 2 (Correctness of Algorithm 2). *Given a free-choice net $N = (P, T, F, A, \sigma)$, a marking $M_0 : P \to \mathbb{N}$, and a transition $t^* \in T$ (with at least two ingoing edges), the result of applying Algorithm 2 on N, M_0, and t^* is a Petri net N' and marking M_0' such that $LTS(N, M_0) \approx LTS(N', M_0')$ and $LTS(N', M_0')$ contains a divergent path iff $LTS(N, M_0)$ contains a divergent path. The transformation time and the increase in size are amortized* $\mathrm{O}(1)$.

Proof. (Sketch, detailed proof in [8].) We define the following relation between pair of markings, and we prove it is a bisimulation:

$$\mathcal{R} := \left\{ \big(M,\ M[p^* \mapsto M(p^*) - i][p^{**} \mapsto M(p^{**}) - i][p^+ \mapsto i]\big) \,\middle|\, \substack{M \in LTS(N, M_0) \\ k = \min(M(p^*), M(p^{**})) \\ 0 \le i \le k} \right\}$$

where M is a reachable marking from the initial marking M_0 in the free-choice net N (M is a state in $LTS(N, M_0)$). The intuition behind the relation is: when $i = 0$, it covers the direct extension of a marking in N to one in N' where the new place p^+ has no tokens; when $i > 0$, it covers the cases where the new transition t^+ has been fired i times without firing t^* afterwards.

For instance, consider Fig. 14 and 15 where $t^* = t_1$, $p^* = p_2$, $p^{**} = p_3$, $t^+ = t_3$ and $p^+ = p_5$. Clearly, the initial markings (M_0, M_0') are in $\mathcal{R}$ since Algorithm 2 just extends M_0 to M_0' where p^+ has zero tokens. Now consider any pair $(M, M') \in \mathcal{R}$. The transition t_2 is not part of the transformed part, so t_2 can be fired in N iff it can be fired in N'. Firing t_1 in N can be replied in N' by firing t_1 when there is a token in p_5 ($i > 0$) and otherwise by firing t_3 followed by t_1 ($i = 0$). Firing t_1 in N' can be replied in N by also firing t_1. If t_3 is fired in N' then N should do nothing as this simply merges the tokens from p_2 and p_3 into one token in p_5 by a τ-action (increases i by one). All these cases result in a new pair in $\mathcal{R}$ which completes the proof of $LTS(N, M_0) \approx LTS(N', M_0')$.

In general, weak bisimulation alone does not ensure the absence of new divergent paths [14]. However, Algorithm 2 only extends existing paths to t^* by one τ-transition such that no divergent paths are changed, so $LTS(N', M_0')$ contains a divergent path iff $LTS(N, M_0)$ contains a divergent path. □

Lemma 1 (Invariant of Algorithm 2). *If Algorithm 2 is given a free-choice net N, it returns a free-choice net N' that has no additional or changed transitions with no ingoing edges compared to N.* (Proof available in [8].)

To achieve our encoding of free-choice workflow into CCS nets we use Algorithm 3, which applies Algorithm 2 (by non-deterministically picking a transition t^*) until all transitions have at most one ingoing edge (or two for τ-transitions). If Algorithm 3 is applied to a free-choice workflow net, it returns a CCS net (Lemma 2) that is weakly bisimilar and has no new divergent paths (Theorem 3).

Algorithm 3: Iterative Petri net transition preset reduction

```
  Input  : Free-choice net $N = (P, T, F, A, \sigma)$ and marking $M_0 : P \to \mathbb{N}$
  Output: Petri net $(P', T', F', A', \sigma')$ and marking $M'_0 : P' \to \mathbb{N}$
1 $(N', M'_0) \leftarrow (N, M_0)$
2 while $\exists t^* \in T'$ where $|\{p \mid (p, t^*) \in F\}| >$ (if $\sigma(t^*) = \tau$ then $2$ else $1$) do
3     $(N', M'_0) \leftarrow$ Algorithm $2(N', M'_0, t^*)$
4 end
5 return $(N', M'_0)$
```

Lemma 2 (Algorithm 3 output). *If applied to a free-choice workflow net (P, T, F, A, σ), Algorithm 3 returns a CCS net.* (Proof: [8].)

Theorem 3 (Correctness of Algorithm 3). *Given a free-choice net $N = (P, T, F, A, \sigma)$ and a marking $M_0 : P \to \mathbb{N}$, the result of applying Algorithm 3 on N and M_0 is a Petri net N' and marking M'_0 such that $LTS(N, M_0) \approx LTS(N', M'_0)$ and $LTS(N, M_0)$ has a divergent path iff $LTS(N', M'_0)$ has. Both the transformation time and the size of (N', M') are $\mathrm{O}(|N|)$.*

Proof. (Sketch, detailed proof in [8].) Follows from induction on the number of applications of Algorithm 2 and transitivity of bisimulation [14]. □

The full encoding from a free-choice workflow net into a weakly bisimilar CCS process is obtained by composing Algorithm 3 and Algorithm 1, leading to Theorem 4. Algorithm 2 can at most be applied $|F|$ times. It runs in constant time and increases the size by a constant amount such that Algorithm 3 produces a linear-sized CCS net. Hence, the resulting CCS process is also linear in size of the original Petri net and marking.

Theorem 4 (Correctness of encoding from free-choice workflow net to CCS). *Let $N = (P, T, F, A, \sigma)$ be a free-choice workflow net and $M_0 : P \to \mathbb{N}$ be a marking. N and M_0 can be encoded into a weakly bisimilar CCS process Q with defining equations $\mathcal{D}$ such that $LTS(N, M_0)$ has a divergent path iff $LTS(Q, \mathcal{D})$ has. The encoding time and the size of $(Q, \mathcal{D})$ are $\mathrm{O}(|N| + \sum_{p \in P} M_0(p))$.*

Proof. By Lemma 1+2, Theorem 3+1 and transitivity of bisimulation [14]. □

4.3 Encoding Any Free-Choice Net into CCS

Although the focus of Section 4.2 is encoding free-choice *workflow* nets into CCS, many definitions and results therein can be applied to *any* free-choice net (Algorithm 2, Theorem 2, Lemma 1, Algorithm 3, Theorem 3). In this section we build upon them to develop an encoding from *any* free-choice net into CCS.

To achieve this result, we define a superclass of CCS nets (Definition 12) called *2-τ-synchronisation nets* (Definition 13, below), which may have transitions with no ingoing edges. Such transitions can be seen as *token generators*: they can always fire and produce tokens; they do not occur in workflow nets,

Algorithm 4: Encoding from 2-τ-synchronisation net to CCS process

```
  Input  : 2-$\tau$-synchronisation net $N = (P, T, F, A, \sigma)$ and marking $M_0 : P \to \mathbb{N}$
  Output: CCS process $Q$ and partial mapping of defining equations $\mathcal{D}$
1 $(\_, \mathcal{D}) \leftarrow$ Algorithm 1$(N, M_0)$
2 $T_0 \leftarrow t \in \{t \mid t \in T$ and $t$ has 0 ingoing edges$\}$
3 for $t \in T_0$ do
4     $\mathcal{D}(X_t) \leftarrow \sigma(t).(X_t \mid X_{p_1} \mid \ldots \mid X_{p_k})$ where $\{p_1, \ldots, p_k\} = \{p \mid (t, p) \in F\}$
5 end
6 $Q \leftarrow (\nu s_1) \ldots (\nu s_n)\Big(X_{p_1}^{M_0(p_1)} \mid \ldots \mid X_{p_{|P|}}^{M_0(p_{|P|})} \mid \underbrace{X_{t_1} \mid \ldots \mid X_{t_k}}_{t_i \in \{t_1, \ldots, t_k\} = T_0}\Big)$
7 return $(Q,\ \mathcal{D})$
```

and they would be dropped if given to Algorithm 1 because they do not originate from any place. For this reason, token generator transitions are handled separately in Algorithm 4 (which extends Algorithm 1).

Definition 13 (2-τ-synchronisation net). *A Petri net (P, T, F, A, σ) is a* 2-τ-synchronisation net *iff each transition $t \in T$ has at most two ingoing edges — and in the latter case, $\sigma(t) = \tau$.*

Line 1 of Algorithm 4 retrieves the defining equations $\mathcal{D}$ returned by Algorithm 1 (the returned CCS process is not used). Line 2 defines T_0 as all transitions $t \in T$ with no ingoing edges. Each transition $t \in T_0$ is encoded as a sequential process named X_t in $\mathcal{D}$ (line 3–5). X_t is defined as $\sigma(t)$ followed by the *parallel composition* of X_t (itself) and all processes named X_{p_i}, where p_i has an ingoing edge from t. This ensures that whenever X_t is used, then it spawns a new copy of itself that can be used again. Finally on line 6, all X_{t_i}, where $t_i \in T_0$, are included once in the process Q to be available from the start. For instance, the simple 2-τ-synchronisation net $N = (\{p_1\}, \{t_1\}, \{(t_1, p_1)\}, \{b\}, [t_1 \mapsto b]\})$ and marking M_0 with $M_0(p_1) = 0$ are encoded into $(X_{t_1}, \mathcal{D})$ where $\mathcal{D}(X_{t_1}) = b.(X_{t_1} \mid X_{p_1})$ which gives a process where X_{t_1} is always present: $X_{t_1} \xrightarrow{b} (X_{p_1} \mid X_{t_1}) \xrightarrow{b} (X_{p_1}^2 \mid X_{t_1}) \xrightarrow{b} \ldots$ Algorithm 4 also produces strongly bisimilar CCS processes like Algorithm 1.

Theorem 5 (Correctness of Algorithm 4). *Given a 2-τ-synchronisation net $N = (P, T, F, A, \sigma)$ and an initial marking $M_0 : P \to \mathbb{N}$, the result of applying Algorithm 4 on N and M_0 is $(Q, \mathcal{D})$ such that $LTS(N, M_0) \sim LTS(Q, \mathcal{D})$. The translation time and the size of $(Q, \mathcal{D})$ are bound by* $\mathrm{O}(|N| + \sum_{p \in P} M_0(p))$.

Proof. (Sketch, detailed proof in [8].) Extend the proof of Theorem 1 by extending the parallel composition in *all* CCS processes in the relation $\mathcal{R}$ with the parallel composition of all X_{t_i}, where $t_i \in T_0$. □

The CCS process obtained from Algorithm 4 is linear in size of the encoded 2-τ-synchronisation net and marking. Theorem 5 allows us to extend the encoding of free-choice workflow nets to free-choice nets.

Lemma 3 (Algorithm 3 output). *If applied to a free-choice net (P, T, F, A, σ), Algorithm 3 returns a 2-τ-synchronisation net.* (Proof: [8].)

Theorem 6 (Correctness of free-choice net to CCS encoding). *Let $N = (P, T, F, A, \sigma)$ be a free-choice net and $M_0 : P \to \mathbb{N}$ be an initial marking. N and M_0 can be encoded into a weakly bisimilar CCS process Q with defining equations $\mathcal{D}$ such that $LTS(N, M_0)$ has a divergent path iff $LTS(Q, \mathcal{D})$ has. The encoding time and the size of $(Q, \mathcal{D})$ are $\mathrm{O}(|N| + \sum_{p \in P} M_0(p))$.*

Proof. By Lemma 1+3, Theorem 3+5 and transitivity of bisimulation [14]. □

4.4 Group-Choice Nets

We now further extend our CCS encodability results. Free-choice nets (Definition 10) only allow for choices taken by a *single* place; we relax this requirement by defining a larger class of nets called *group-choice nets* (Definition 15, below), where choices can be taken by a *group* of places which synchronise before making the choice. This requires that all places in the group *agree* on their options — which corresponds to all places in the group having the same postset (i.e., edges to the same transitions). No other places outside the group may have edges to those transitions (as that would affect the choice).

Definition 14 (Postset of a place). *Let (P, T, F, A, σ) be a Petri net. The* postset *of a place $p \in P$ (written $p\bullet$) is the set of all transitions with an ingoing edge from p, i.e.: $p\bullet := \{t \,|\, p \in P, (p, t) \in F\}$.*

Definition 15 (Group-choice net). *A Petri net N is a* group-choice net *iff all pairs of places either have a same postset, or disjoint postsets.*

A group-choice net does not have partially overlapping place postsets. Therefore, its places can be split into *disjoint groups* where all places in a group have the same postset. Fig. 16–18 show examples of Petri nets and postsets for each place. Fig. 16 is not a group-choice net because $p_1\bullet$ and $p_2\bullet$ are only partially overlapping (since p_2 has an edge to t_2 while p_1 does not). Fig. 17 is a group-choice net, hence its places can be split into two disjoint groups $\{p_1, p_2\}$ and $\{p_3\}$. Fig. 18 is a group-choice net with three groups of places.

A place postset of size at least two (like $p_4\bullet$ in Fig. 18) corresponds to a choice because the place can choose between at least two transitions. When at

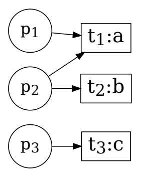


Fig. 16. Petri net with $p_1\bullet = \{t_1\}$, $p_2\bullet = \{t_1, t_2\}$ and $p_3\bullet = \{t_3\}$.

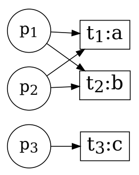


Fig. 17. Group-choice net with $p_1\bullet = p_2\bullet = \{t_1, t_2\}$ and $p_3\bullet = \{t_3\}$.

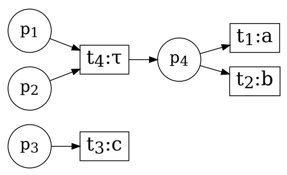


Fig. 18. Result of Algorithm 5 on Fig. 17. $p_1\bullet = p_2\bullet = \{t_4\}$, $p_3\bullet = \{t_3\}$ and $p_4\bullet = \{t_1, t_2\}$.

Algorithm 5: Group-choice net transition preset reduction

```
Input  : Petri net $N = (P, T, F, A, \sigma)$, marking $M_0 : P \to \mathbb{N}$ and two selected
         places $p^*, p^{**} \in P$ where $p^*\bullet = p^{**}\bullet \neq \emptyset$
Output: Petri net $(P', T', F', A', \sigma')$ and marking $M'_0 : P' \to \mathbb{N}$
1 $P' \leftarrow P \cup \{p^+\}$ where $p^+ \notin P \cup T$
2 $T' \leftarrow T \cup \{t^+\}$ where $t^+ \notin P' \cup T$
3 $F' \leftarrow \left(\begin{array}{l}\left(F \setminus \left(\{(p^*, t) \mid t \in p^*\bullet\} \cup \{(p^{**}, t) \mid t \in p^{**}\bullet\}\right)\right) \\ \quad \cup \ \{(p^*, t^+), (p^{**}, t^+), (t^+, p^+)\} \ \cup \ \{(p^+, t) \mid t \in p^*\bullet\}\end{array}\right)$
4 $N' \leftarrow (P', T', F', A \cup \{\tau\}, \sigma[t^+ \mapsto \tau])$
5 return $(N', M_0[p^+ \mapsto 0])$
```

least two places have the same postset (like $p_1\bullet$ and $p_2\bullet$ in Fig. 18), it corresponds to synchronisation because the transitions in the postset have at least two ingoing edges. The definition of group-choice nets ensures that choice and synchronisation can only happen at the same time if the group of places can safely synchronise before making a choice together. A free-choice net further require that every place postset must not correspond to both choice and synchronisation.

The transformation of group-choice nets into 2-τ-synchronisation nets is intuitively similar to the transformation of free-choice nets (in Section 4.2). The key difference is that, since places in group-choice nets can have more than one outgoing edge when they synchronise on a transition, we consider whole place postsets instead of a single transition. Algorithm 5 does this by selecting two places p^* and p^{**} with the same (non-empty) postset: It removes all their outgoing edges (line 3) and connects them to a new τ-transition t^+ (defined on line 2) with an edge to a new place p^+ (defined on line 1) that has outgoing edges to the same postset p^* and p^{**} originally had (lines 3–5). This reduces the number of ingoing edges (i.e., the *preset*) of *all* transitions in $p^*\bullet$ by one. Fig. 18 shows the result of one application of Algorithm 5 on Fig. 17 with $p^* = p_1$ and $p^{**} = p_2$.

Algorithm 6 keeps applying Algorithm 5 on the input group-choice net (in the same fashion as Algorithm 3) by non-deterministically picking p_1 and p_2, until a 2-τ-synchronisation net is obtained. All the proofs for the encoding of group-choice nets are similar to the ones for free-choice nets: The only difference

Algorithm 6: Iterative group-choice net transition preset reduction

```
Input  : Group-choice net $N = (P, T, F, A, \sigma)$ and marking $M_0 : P \to \mathbb{N}$
Output: Petri net $(P', T', F', A', \sigma')$ and marking $M'_0 : P' \to \mathbb{N}$
1 $(N', M'_0) \leftarrow (N, M_0)$
2 while $\exists t^* \in T'$ where $|\{p \mid (p, t^*) \in F\}| >$ (if $\sigma(t^*) = \tau$ then 2 else 1) do
3     select $(p_1, t^*), (p_2, t^*) \in F$ where $p_1 \neq p_2$
4     $(N', M'_0) \leftarrow$ Algorithm 5$(N', M'_0, p_1, p_2)$
5 end
6 return $(N', M'_0)$
```

is that the proofs for group-choice nets consider *sets* of impacted transitions instead of single transitions. Therefore, we only present the main result below.

Theorem 7 (Correctness of group-choice net to CCS encoding). *A group-choice net $N = (P, T, F, A, \sigma)$ and an initial marking $M_0 : P \to \mathbb{N}$ can be encoded into a weakly bisimilar CCS process Q with defining equations $\mathcal{D}$ s.t. $LTS(N, M_0)$ has a divergent path iff $LTS(Q, \mathcal{D})$ has. The encoding time and the size of $(Q, \mathcal{D})$ are $\mathrm{O}(|N| + \sum_{p \in P} M_0(p))$.* (Proofs in [8].)

5 Conclusion and Future Work

In this paper we have formalised a direct encoding of CCS nets into strongly bisimilar CCS processes (akin to [13]), and on this foundation we have developed novel Petri net-to-CCS encodings and results, through a series of generalizations. We have introduced an encoding of *free-choice workflow nets* into weakly bisimilar CCS processes. We have generalized this result as an encoding from *any* free-choice net into a weakly bisimilar CCS processes, via a new superclass of CCS nets called *2-τ-synchronisation nets* (which may have token-generating transitions). Finally, we have presented a superclass of free-choice nets called *group-choice nets*, and its encoding into weakly bisimilar CCS processes.

On the practical side, we are now exploring the practical application of these results — e.g. using model checkers oriented towards process calculi (such as mCRL2 [9]) to analyse the properties of α-mined Petri nets encoded into CCS. On the theoretical side, we are studying how to encode even larger classes of Petri nets into weakly bisimilar CCS processes. One of the problems is how to handle partially-overlapping place postsets: in this paper, we focused on classes of Petri nets without such overlaps, and 2-τ synchronisation nets. Another avenue we are exploring is to develop further encodings after dividing a Petri net into groups of places/transitions, as we did for group-choice nets.

Acknowledgements. We wish to thank to the anonymous reviewers for their comments and suggestions, and Ekkart Kindler for the fruitful discussions.

References

1. van der Aalst, W.M.P.: Pi calculus versus petri nets: Let us eat "humble pie" rather than further inflate the "pi hype". BPTrends **3** (01 2005)
2. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016). https://doi.org/10.1007/978-3-662-49851-4
3. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004). https://doi.org/10.1109/TKDE.2004.47
4. Aranda, J., Valencia, F.D., Versari, C.: On the expressive power of restriction and priorities in CCS with replication. In: de Alfaro, L. (ed.) Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS

2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5504, pp. 242–256. Springer (2009). https://doi.org/10.1007/978-3-642-00596-1_18

5. Baldan, P., Bonchi, F., Gadducci, F.: Encoding asynchronous interactions using open petri nets. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009 - Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5710, pp. 99–114. Springer (2009). https://doi.org/10.1007/978-3-642-04081-8_8
6. Baldan, P., Bonchi, F., Gadducci, F., Monreale, G.V.: Encoding synchronous interactions using labelled petri nets. In: Kühn, E., Pugliese, R. (eds.) Coordination Models and Languages - 16th IFIP WG 6.1 International Conference, COORDINATION 2014, Held as Part of the 9th International Federated Conferences on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014, Proceedings. Lecture Notes in Computer Science, vol. 8459, pp. 1–16. Springer (2014). https://doi.org/10.1007/978-3-662-43376-8_1
7. Bogø, B.: Encoding petri nets into ccs (Mar 2024). https://doi.org/10.5281/zenodo.10855866, https://dtu.bogoe.eu/tools/pn2ccs/
8. Bogø, B., Burattin, A., Scalas, A.: Encoding petri nets into ccs (technical report) (2024). https://doi.org/10.48550/arXiv.2404.14385
9. Bunte, O., Groote, J.F., Keiren, J.J.A., Laveaux, M., Neele, T., de Vink, E.P., Wesselink, W., Wijs, A., Willemse, T.A.C.: The mcrl2 toolset for analysing concurrent systems - improvements in expressivity and usability. In: Vojnar, T., Zhang, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11428, pp. 21–39. Springer (2019). https://doi.org/10.1007/978-3-030-17465-1_2
10. Desel, J., Esparza, J.: Free choice Petri nets. No. 40, Cambridge university press (1995)
11. van Dongen, B.F., de Medeiros, A.K.A., Wen, L.: Process mining: Overview and outlook of petri net discovery algorithms. Trans. Petri Nets Other Model. Concurr. **2**, 225–242 (2009). https://doi.org/10.1007/978-3-642-00899-3_13
12. Goltz, U.: CCS and petri nets. In: Guessarian, I. (ed.) Semantics of Systems of Concurrent Processes, LITP Spring School on Theoretical Computer Science, La Roche Posay, France, April 23-27, 1990, Proceedings. Lecture Notes in Computer Science, vol. 469, pp. 334–357. Springer (1990). https://doi.org/10.1007/3-540-53479-2_14
13. Gorrieri, R., Versari, C.: A process calculus for expressing finite place/transition petri nets. In: Fröschle, S.B., Valencia, F.D. (eds.) Proceedings 17th International Workshop on Expressiveness in Concurrency, EXPRESS'10, Paris, France, August 30th, 2010. EPTCS, vol. 41, pp. 76–90 (2010). https://doi.org/10.4204/EPTCS.41.6
14. Gorrieri, R., Versari, C.: Introduction to Concurrency Theory - Transition Systems and CCS. Texts in Theoretical Computer Science. An EATCS Series, Springer (2015). https://doi.org/10.1007/978-3-319-21491-7
15. Meyer, R., Khomenko, V., Hüchting, R.: A polynomial translation of π-calculus (FCP) to safe petri nets. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7454, pp. 440–455. Springer (2012). https://doi.org/10.1007/978-3-642-32940-1_31

16. Stefansen, C.: SMAWL: A small workflow language based on CCS. In: Belo, O., Eder, J., e Cunha, J.F., Pastor, O. (eds.) The 17th Conference on Advanced Information Systems Engineering (CAiSE '05), Porto, Portugal, 13-17 June, 2005, CAiSE Forum, Short Paper Proceedings. CEUR Workshop Proceedings, vol. 161. CEUR-WS.org (2005)