PODPIS ZAUFANY
GRZEGORZ
BAZYDŁO
18.09.2025 23:46:44 [GMT+2]
Dokument podpisany elektronicznie
podpisem zaufanym

PODPIS ZAUFANY
MATEUSZ PAWEŁ
POPŁAWSKI
19.09.2025 08:36:03 [GMT+2]
Dokument podpisany elektronicznie
podpisem zaufanym

UNIWERSYTET ZIELONOGÓRSKI

DZIEDZINA NAUK INŻYNIERYJNO-TECHNICZNYCH

DYSCYPLINA: INFORMATYKA TECHNICZNA I TELEKOMUNIKACJA

ROZPRAWA DOKTORSKA

Analiza żywotności części sterującej systemu cyber-fizycznego specyfikowanej z zastosowaniem sieci Petriego

Liveness analysis of the control part of a cyber-physical system specified by a Petri net

mgr inż. Mateusz Popławski

Opieka naukowa nad przygotowaniem rozprawy doktorskiej:

Promotor:
prof. dr hab. inż. Remigiusz Wiśniewski

Promotor pomocniczy:
dr inż. Grzegorz Bazydło

Rozprawę akceptuję:

Signed by / Podpisano przez:
.....
Remigiusz Grzegorz
Wiśniewski
.....
Date / Data: 2025-09-18 22:
28

Zielona Góra, wrzesień 2025

Autor pragnie bardzo serdecznie podziękować promotorowi prof. dr hab. inż. Remigiuszowi Wiśniewskiemu oraz promotorowi pomocniczemu dr inż. Grzegorzowi Bazydło za ich cierpliwość, przewodnictwo oraz konstruktywne sugestie. Poza tym chciałbym podziękować mojemu koledze dr inż. Marcinowi Wojnakowskiemu, który udzielił wielu cennych porad dotyczących pracy nad rozprawą.

Rozprawę dedykuję mojej ukochanej narzeczonej

Badania zamieszczone w pracy wykonano w ramach projektu badawczego OPUS realizowanego w latach 2020-2024 (temat projektu: „Analiza oraz dekompozycja części sterującej systemu cyber-fizycznego opisanego z zastosowaniem interpretowanej sieci Petriego", nr projektu 2019/35/B/ST6/01683, kierownik projektu: prof. dr hab. inż. Remigiusz Wiśniewski) finansowanego ze środków Narodowego Centrum Nauki.

Streszczenie

W pracy doktorskiej zaproponowana została metoda weryfikacji żywotności części sterującej systemów cyber-fizycznych modelowanych przy pomocy sieci Petriego. Na początku dokonano wprowadzenia do tematyki systemów cyber-fizycznych oraz scharakteryzowano główne obszary ich zastosowań. Przedstawiono także wybrane zagadnienia z teorii grafów oraz sieci Petriego, aby szczegółowo wyjaśnić sposób działania zaproponowanej metody badania żywotności. Opisano również w jaki sposób można dokonywać szacowania złożoności obliczeniowej, co umożliwiło obiektywne porównanie innych algorytmów oraz metod analizy z proponowanym rozwiązaniem. Dokonano przeglądu technik weryfikacji żywotności, a także metod analizujących inne właściwości sieci (np. ograniczoność), ale powiązanych z analizą żywotności. Wybrano oraz szczegółowo opisano metodę referencyjną, do której porównano autorskie rozwiązanie analizy żywotności części sterującej systemu cyber-fizycznego. W pracy opisano również proces implementacji zaproponowanych metod, a także wykonano badania eksperymentalne autorskiego rozwiązania oraz metody referencyjnej na dedykowanym sprzęcie komputerowym. Na podstawie przeprowadzonych badań eksperymentalnych potwierdzono efektywność oraz skuteczność proponowanej techniki analizy żywotności. Ponadto, przeprowadzono dyskusję uzyskanych wyników badań oraz za pomocą wybranych przykładów zaprezentowano korzyści oraz ograniczenia proponowanego rozwiązania. Wskazano też perspektywy rozwoju opracowanego podejścia oraz nakreślono kierunki dalszych prac.

Słowa kluczowe: sieć Petriego, żywotność, analiza żywotności, system cyber-fizyczny, część sterująca systemu cyber-fizycznego.

Abstract

The doctoral thesis focuses on the liveness analysis of the control part of cyber-physical systems modeled by Petri nets. In particular, a novel liveness analysis technique is proposed and described. Firstly, an introduction to cyber-physical systems was made and the main areas of their applications were briefly summarized. The selected issues from the graph and Petri net theories are also presented in order to explain the proposed technique. Furthermore, the estimations of algorithms' computational complexity are presented in regard to the comparison of the other algorithms and analysis methods with the proposed solution. Next, existing analysis techniques and methods related to the other Petri net properties (e.g., boundedness) are reviewed. Moreover, a reference method is selected and described in detail. Finally, a description of the implementation process is provided, as well as the results of experimental research. The obtained research results are discussed in detail and compared with the reference method in order to show the effectiveness and efficiency of the technique proposed in the dissertation. Using examples, the benefits and limitations of the proposed solution were presented. The proposed approach's development prospects and future research directions were also indicated.

Keywords: Petri net, liveness, liveness analysis, cyber-physical system, control part of the cyber-physical system.

Spis treści

Spis najważniejszych skrótów i symboli

CAD	**Computer Aided Design**
CPS	Cyber-Physical System
FBD	Function Block Diagram
FPGA	Field-Programmable Gate Array
IL	Instruction List
LD	Ladder Diagram
PLC	Programmable Logic Controller
PNH	Petri Net Hippo
SFC	Sequential Function Chart
SMC	State-Machine Component
ST	Structured Text
$\mathbb{Z}_+$	Zbiór liczb całkowitych nieujemnych
$\mathbb{N}$	Zbiór liczb naturalnych
$\emptyset$	Zbiór pusty
$A \cup B$	Suma zbiorów A i B
$A \cap B$	Część wspólna zbiorów A i B
$A \subseteq B$	Zbiór A jest podzbiorem zbioru B
$x \in X$	x jest elementem zbioru X
$A \times B$	Iloczyn Kartezjański zbiorów A i B
$\exists$	Istnieje
$\forall$	Dla każdego

1. Wstęp

W ostatnich latach następuje bardzo dynamiczny rozwój oraz integracja systemów informatycznych, systemów wbudowanych, czy systemów programowo-sprzętowych. Ich celem jest usprawnianie pracy w różnych branżach, optymalizacja dostępnych zasobów (np. energii elektrycznej, ciepła do ogrzewania domów i mieszkań), automatyzacja procesów przemysłowych oraz biznesowych, wsparcie w dostępie do przestrzeni publicznych, jak również zwiększenie komfortu i bezpieczeństwa domowników. Systemy wykorzystywane w ww. obszarach często określane są rozwiązaniami inteligentnymi (ang. *smart solutions, smart systems*). W literaturze [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] można odnaleźć informacje, że rozwiązania inteligentne występują w bardzo wielu obszarach, w tym między innymi w bankowości, branży produkcyjnej, motoryzacji i transporcie, górnictwie, opiece medycznej, a także w systemach obecnych w inteligentnych miastach (ang. *smart cities*), budynkach (ang. *smart buildings*) czy domach (ang. *smart homes*). Jak nietrudno zauważyć, tego typu rozwiązania wpływają bezpośrednio jak również pośrednio na życie człowieka w różnych jego aspektach.

Warto podkreślić, że omawiane systemy zazwyczaj składają się z dwóch części – jedna z nich odpowiada za sterowanie i przetwarzanie danych (część „cyber"), a druga za fizyczną realizację zadań (część „fizyczna"). W takim podejściu system może zostać określony jako *system cyber-fizyczny* (ang. c*yber-physical system* – CPS[1]) [19]. Systemy cyber-fizyczne zaliczają się do klasy systemów współbieżnych, zarówno pod względem sterowania, jak również wykonywania zadań [15], [16], [17], [18], [19].

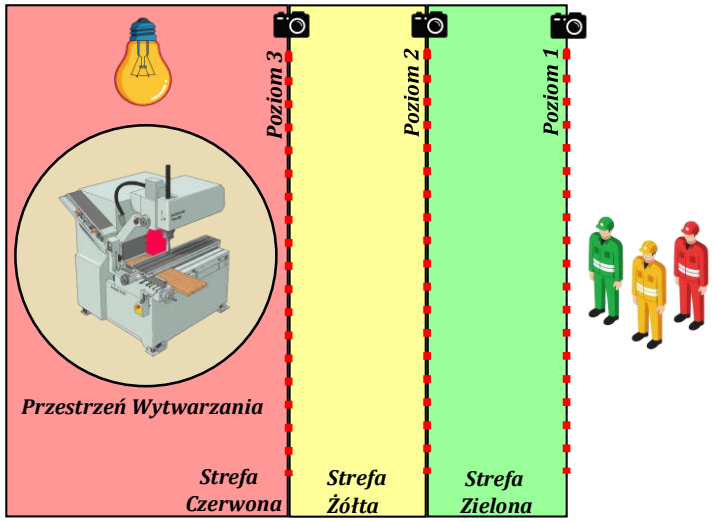
Wobec ogromnego postępu technologicznego ostatnich lat złożoność systemów CPS bardzo wzrosła. Jeszcze kilka lat temu takie systemy składały się z niewielu elementów, co skutkowało stosunkowo małą liczbą sygnałów wejściowych i wyjściowych oraz sprawiało, że projektowanie oraz weryfikacja części sterującej nie były skomplikowane (w przypadku prostych systemów można było wykonać to nawet ręcznie, bez specjalistycznego oprogramowania wspomagającego). Obecnie zaprojektowanie oraz zweryfikowanie poprawności części sterującej bez pomocy dedykowanych narzędzi jest bardzo trudne i podatne na błędy ludzkie, a często wręcz niemożliwe. Takie dedykowane narzędzia realizowane są zazwyczaj w formie oprogramowania komputerowego, które kwalifikowane jest jako narzędzie do projektowania wspomaganego komputerowo (ang.

[1] w niniejszej pracy termin „system cyber-fizyczny" będzie zamiennie stosowany z terminem „system CPS"

Computer Aided Design – CAD). W oprogramowaniu klasy CAD wykorzystuje się różnego rodzaju techniki projektowania (modelowania), między innymi:

- automaty skończone (ang. *Finite State Machine* – FSM) [20], [21],
- sieci Petriego (w szczególności interpretowane sieci Petriego) [17], [22], [23], [24],
- diagramy stanów (np. diagramy statechart, diagramy maszyn stanowych z języka UML).

Przykładem systemu CPS, który wymaga zaprojektowania i weryfikacji ze wsparciem oprogramowania w celu uniknięcia potencjalnych błędów, które mogą mieć daleko idące skutki (np. zagrożenie życia i zdrowia operatorów systemu) jest system produkcyjny z trzema strefami bezpieczeństwa i komputerowym systemem wizyjnym, którego schemat koncepcyjny przedstawiono na rysunku 1.1 [17], [25]. Ma on za zadanie sterowanie procesem wycinania wymaganych elementów z drewna na frezarce oraz kontrolę dostępu do stref przy samej frezarce. W razie wykrycia naruszenia danej strefy przez osoby nie posiadające uprawnień (w przykładzie występują trzy strefy oznaczone różnymi kolorami i każda z nich ma odrębne zasady dostępu) system uruchamia alarm, a w razie konieczności zatrzymuje działanie frezarki oraz systemu podawania drewna.



Rysunek 1.1. Schemat koncepcyjny systemu produkcyjnego z trzema strefami i komputerowym systemem wizyjnym [17], [25]

Na postawie zaprezentowanego przykładu można wyciągnąć pewne istotne wnioski. Po pierwsze, bardzo ważnym aspektem w procesie projektowania systemu CPS oprócz wydajnego działania systemu jest również bezpieczeństwo osób, które mogą znaleźć się przy systemie działającym produkcyjnie. Brak błędów na etapie projektowania, weryfikacji czy symulacji działania systemu może być kluczowe, aby system działał wydajnie, realizował przydzielone mu zadania, ale również był bezpieczny dla operatora systemu

oraz osób, które mogą znaleźć się w jego otoczeniu. Wykorzystanie oprogramowania oraz narzędzi, które wspomagają proces projektowania, weryfikacji i symulacji minimalizuje ryzyko popełnienia błędów już na etapie specyfikacji systemu.

1.1. Motywacja podjęcia tematu

Jak przedstawiono w poprzednim podrozdziale, istnieje zapotrzebowanie na efektywne modelowanie i weryfikację systemów cyber-fizycznych (zwłaszcza jego części sterującej). Modelowanie z jednej strony powinno wspierać wysoki poziom abstrakcji (zwłaszcza pod kątem dalszych etapów projektowania systemu), a z drugiej dawać możliwość weryfikacji poprawności części sterującej CPS. Sieci Petriego spełniają obie wyżej wymienione cechy. Wykorzystanie sieci Petriego umożliwia zaprojektowanie części sterującej systemu cyber-fizycznego za pomocą wysoko abstrakcyjnych elementów, które są reprezentowane symbolami graficznymi takimi jak okrąg (miejsce), prostokąt (tranzycja) oraz strzałka (łuk), jak również pozwala otrzymać formalny opis takiej sieci (specyfikacja) [19], [26]. Ponadto sieci Petriego są również wspierane przez formalne mechanizmy, które umożliwiają analizę modelu [27], [28], [29], [30]. Wykonanie analizy części sterującej systemu CPS pozwala na wykrycie błędów już na etapie specyfikowania systemu co może wpłynąć na redukcję czasu oraz kosztów naprawy takich błędów [17]. Wykrycie błędu i próba jego naprawienia na dalszych etapach realizacji systemu może być bardzo kosztowne (czasowo oraz finansowo), a w szczególnych przypadkach koszty tej naprawy mogą przewyższać przygotowanie systemu od początku.

W niniejszej rozprawie proponowane jest zastosowanie interpretowanych sieci Petriego. Jest to swoiste rozszerzenie klasycznych sieci Petriego, przy czym interpretowana sieć Petriego posiada dodatkowe własności. Przede wszystkim, sieć interpretowana jest wyposażona w sygnały wejściowe oraz wyjściowe, umożliwiające komunikację systemu z otoczeniem. Jest to szczególnie istotne w przypadku części sterującej systemu CPS, gdyż za pomocą sygnałów wejściowych oraz wyjściowych istnieje możliwość sterowania (zarządzania) częścią fizyczną systemu. Ponadto, interpretowana sieć Petriego powinna w założeniu spełniać dwie najistotniejsze własności: być żywa oraz ograniczona (w niektórych przypadkach ten drugi warunek może być bardziej restrykcyjny, wówczas wymagane jest, aby sieć była nie tylko ograniczona, ale i bezpieczna).

Aktualnie wielu naukowców na świecie prowadzi badania, których celem jest efektywne specyfikowanie części sterującej systemów cyber-fizycznych z wykorzystaniem

interpretowanych sieci Petriego. Można jednak napotkać pewne problemy przy analizie systemów opartych o sieci interpretowane, a dokładniej przy weryfikacji czy analizowana sieć posiada konkretne właściwości, zwłaszcza żywotność oraz ograniczoność (które są wymagane z definicji interpretowanej sieci Petriego). Metody weryfikacji powyższych właściwości mogą zostać ogólnie podzielone na dwie zasadnicze grupy. Pierwsza z nich wykorzystuje analizę przestrzeni stanów systemu, a techniki z drugiej grupy bazują na algebrze liniowej [29], [31], [32], [33], [34]. W przypadku metod z pierwszej grupy problematyczny może się okazać wykładniczy przyrost stanów analizowanej przestrzeni – taką sytuację nazywa się *problemem eksplozji stanów* (ang. *state explosion problem*) [27]. Metody z drugiej grupy też nie są wolne od problemu wykładniczości i w procesie obliczania kolejnych wartości ich liczba może rosnąć wykładniczo [33], [35], [36]. Reasumując, koncepcje z obu grup w ogólnym przypadku borykają się z problemem wykładniczej złożoności obliczeniowej, co może skutkować nieuzyskaniem wyniku w zadanym (akceptowalnym) czasie, a to z kolei oznacza, że metody te mogą zostać uznane za nieskuteczne (w uproszczeniu pojęcie skuteczności wiąże się z poprawnością uzyskanych rezultatów, a formalnie zostanie ono wprowadzone w dalszej części rozprawy) [33].

Jak już wspomniano, jedną z dwóch najważniejszych właściwości kluczowych w procesie weryfikacji poprawności modelu części sterującej systemu CPS opisanego siecią Petriego jest żywotność. To właśnie między innymi ta właściwość pozwala określić, czy każda zamodelowana operacja (zaprojektowany element) ma szanse się uruchomić. Innymi słowy, pozwala stwierdzić, czy nie wystąpi tu sytuacja, że jakiś fragment (proces) w modelowanym systemie w trakcie jego działania nigdy nie zostanie użyty. W połączeniu z właściwościami takimi jak ograniczoność (czy bardziej restrykcyjnie – bezpieczeństwo) modelowany system można uznać za poprawny. Przy czym należy tu podkreślić, że poprawność odnosi się ściśle do aspektów weryfikacji modelu systemu, bez przypisanych sygnałów sterujących. Nieodpowiednie przypisanie sygnałów sterujących, nawet w przypadku poprawnego modelu systemu może spowodować jego niepoprawne działanie, lecz zagadnienia te wykraczają poza zakres niniejszej rozprawy.

1.2. Teza, cele i zakres pracy

Kierując się motywacjami z poprzedniego rozdziału przeprowadzono badania, na podstawie których sformułowano następującą tezę niniejszej pracy doktorskiej:

Możliwe jest przeprowadzenie analizy żywotności części sterującej systemu cyber-fizycznego specyfikowanej z zastosowaniem sieci Petriego w sposób efektywny oraz skuteczny.

Celem pracy jest **opracowanie efektywnej** i **skutecznej metody analizy żywotności części sterującej systemu cyber-fizycznego specyfikowanej z zastosowaniem sieci Petriego**. W szczególności, opracowana metoda składać się będzie z poszczególnych *technik* (metod i algorytmów), stanowiących zarówno koncepcje nowatorskie, jak i będące modyfikacją (usprawnieniem) metod już istniejących. W pracy przyjęto, że *efektywność* odnosi się do czasu wykonania algorytmu (tzn. ukończenia działania algorytmu w zadanym czasie), z kolei *skuteczność* oznacza uzyskanie poprawnego wyniku.

W celu udowodnienia tezy wyznaczono następujące zadania:

- analiza istniejących metod analizy żywotności systemu opisanego siecią Petriego;
- zaproponowanie nowych oraz usprawnienie już istniejących **efektywnych** oraz **skutecznych** technik analizy żywotności części sterującej systemu cyber-fizycznego opisanej siecią Petriego (w tym przypadku pod pojęciem „technika" rozumiana jest pojedyncza metoda wspierająca proces analizy żywotności systemu);
- **opracowanie kompletnej metody** umożliwiającej analizę żywotności systemu cyber-fizycznego opisanego siecią Petriego (w szczególności metoda składać się będzie z poszczególnych technik analizy żywotności);
- implementacja programowa opracowanych technik oraz kompletnej metody analizy żywotności części sterującej systemu CPS opisanej siecią Petriego;
- badania eksperymentalne ukierunkowane na sprawdzenie efektywności oraz sprawności opracowanej metody (w tym opracowanych technik) analizy żywotności części sterującej systemu cyber-fizycznego opisanej siecią Petriego.

Należy podkreślić, że wymiernym efektem niniejszej pracy są fizyczne metody i programy, zaimplementowane w ramach sytemu *Hippo* [32], [37], [38], [39]. System *Hippo* (*www.hippo.uz.zgora.pl*) jest wynikiem prac zespołu badawczego pod kierunkiem prof. dr hab. inż. R. Wiśniewskiego i jest intensywnie rozwijany na przestrzeni ostatnich lat.

Ponadto założono, że badanie efektywności oraz skuteczności opracowanych technik zostanie potwierdzone zarówno teoretycznie jak i (głównie) eksperymentalnie:

- teoretycznie: dla wybranych technik, charakteryzujących się wielomianową złożonością obliczeniową, oszacowana zostanie złożoność obliczeniowa danego algorytmu;
- eksperymentalnie: dla zbioru 246 systemów opisanych siecią Petriego (wszystkich dostępnych w systemie *Hippo* na moment pisania niniejszej rozprawy) przeprowadzone zostaną szczegółowe badania eksperymentalne pod kątem sprawdzenia efektywności i skuteczności opracowanej metody analizy żywotności; w tym celu wyniki osiągnięte z zastosowaniem proponowanej metody zostaną porównanie z rezultatami uzyskanymi dla metody referencyjnej (klasycznej).

1.3. Struktura pracy

Praca została podzielona na siedem rozdziałów. W pierwszym rozdziale przedstawiono wprowadzenie do problematyki, sformułowano tezę oraz motywację podjęcia tematu. W drugim rozdziale wprowadzono oraz wyjaśniono zagadnienia dotyczące systemów cyber-fizycznych. Również w tym rozdziale wprowadzono zagadnienia teoretyczne związane z sieciami Petriego, między innymi niezbędne definicje z teorii grafów oraz notacje i definicje z teorii sieci Petriego. W dalszej części rozdziału opisano zagadnienia związane ze złożonością obliczeniową oraz przeprowadzono przegląd dostępnych metod analizy sieci Petriego (pod kątem weryfikacji żywotności). Trzeci rozdział został przeznaczony na opis metody referencyjnej weryfikacji żywotności, w którym szczegółowo opisano kolejne etapy działania tej metody. Przedstawiono w nim kolejne etapy działania metody referencyjnej wraz ze szczegółowym opisem. W czwartym rozdziale przedstawiono zaproponowaną przez autora metodę analizy żywotności, która składa się z wielu technik, a w następnym rozdziale zaprezentowano wyniki badań eksperymentalnych opracowanej metody. W rozdziale szóstym zaprezentowano wybrane przykłady (z bazy testowej *Hippo*) ilustrujące zastosowanie opracowanej metody. W ostatnim rozdziale podsumowano rozprawę oraz wskazano plany dalszych badań. Każdy z rozdziałów (z wyjątkiem ostatniego oraz wstępu) posiada także krótkie podsumowanie.

2. Analiza części sterującej systemu cyber-fizycznego specyfikowanej z zastosowaniem sieci Petriego

W rozdziale przedstawiono definicję systemu cyber-fizycznego (ang. *Cyber-Physical System* – CPS) oraz główne obszary zastosowań tego typu systemów. Nakreślono także wyzwania stojące przed projektantami systemów CPS, zwłaszcza dot. modelowania i implementacji części sterującej tych systemów. W dalszej części rozdziału opisano czym są sieci Petriego oraz w jakich dziedzinach są wykorzystywane. Wskazano też zalety wykorzystania ich w procesie modelowania (specyfikacji) oraz analizy części sterującej systemów CPS. W rozdziale nie zabrakło również wprowadzenia teoretycznego (niezbędne definicje) dot. sieci Petriego oraz teorii grafów, na których oparte są sieci Petriego. Dokonano też przeglądu wybranych właściwości sieci Petriego (bezpośrednio powiązanych z żywotnością sieci Petriego, która jest głównym tematem pracy) oraz opisano najczęściej wykorzystywane metody ich weryfikacji. W tym rozdziale wprowadzono również zagadnienia dotyczące złożoności obliczeniowej algorytmów.

2.1. Systemy cyber-fizyczne (CPS)

System cyber-fizyczny (CPS) to połączenie części sterującej (część „cyber") z fizycznymi elementami sytemu (część „fizyczna"). Cześć sterująca („cyber") realizowana jest często przy pomocy komputerów lub układów wbudowanych [15], [16], [25], [32], [39], [40], [41]. Głównym zadaniem części sterującej jest nadzorowanie oraz kontrola fizycznych procesów. W trakcie działania systemu zbierane są również informacje zwrotne (np. z czujników oraz sensorów), które mogą wpływać na obliczenia w części sterującej, na podstawie których kontrolowana jest część fizyczna.

W ostatnich latach prężny rozwój systemów CPS nastąpił m.in. w obszarach, takich jak: motoryzacja, lotnictwo, produkcja, medycyna, gospodarka wodna, transport, kontrola dostępu oraz monitoring, systemy wojskowe, systemy inteligentne (np. systemy klasy *smart home*, *smart buliding*, *smart city* itp.).

Bardzo dużym wyzwaniem przy projektowaniu i wdrażaniu systemów CPS jest łączenie modeli oraz metod inżynierskich z różnych dziedzin z modelami i metodami informatycznymi. Naukowcy w swoich pracach [42], [43] podkreślają, że ze względu na bardzo dużą złożoność takich systemów powinny być one traktowane jako odrębna dyscyplina inżynierska, która wymaga własnych modeli, metod oraz technik weryfikacji poprawności. Sam termin „system cyber-fizyczny" w literaturze pojawił się w okolicach

roku 2006 i został on wprowadzony przez Helen Gill – członkinię National Science Foundation [19]. Terminologia dot. systemów cyber-fizycznych łączona jest również z innymi zagadnieniami takimi jak np. Internet of Things [44], [45], [46], [47], [48], Przemysł 4.0 [22], [49], [50], [51], Internet przemysłowy [52], [53], Machine-to-Machine [54], [55], [56], Internet of Everything [57].

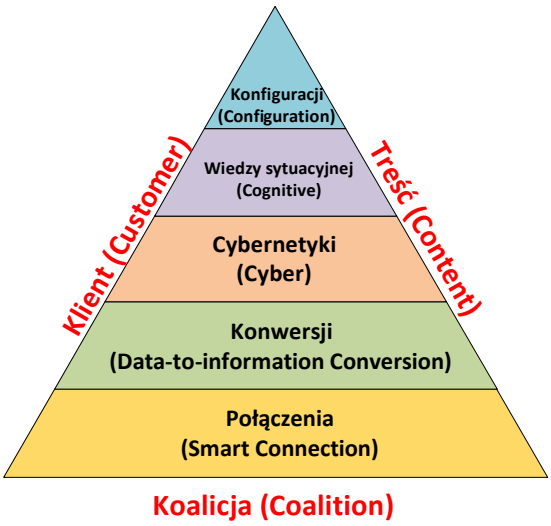
Architektura systemów cyber-fizycznych nie doczekała się jeszcze wspólnego, powszechnego standardu, jednak w literaturze można odnaleźć kilka propozycji architektury takich systemów, np. ISA-95, CPS 5C oraz CPS 8C. Architektura ISA-95 opracowana została przez Amerykański Narodowy Instytut Normalizacyjny w ramach standardu IEC/ISO 62264. Architektura ISA-95 składa się z pięciu warstw [58]:

- warstwa 0 – definiuje rzeczywiste procesy fizyczne,
- warstwa 1 – definiuje czynności związane z wykrywaniem i manipulowaniem procesami fizycznymi,
- warstwa 2 – definiuje czynności monitorowania i sterowania procesami fizycznymi,
- warstwa 3 – definiuje czynności przepływu pracy w celu wytworzenia pożądanych produktów końcowych,
- warstwa 4 – definiuje działania związane z biznesem potrzebne do zarządzania operacją produkcyjną.

Architektura CPS 5C składa się z pięciu warstw: połączenia, konwersji, cybernetycznej, wiedzy sytuacyjnej i konfiguracji [50]. Architektura CPS 8C jest rozwinięciem architektury CPS 5C, do której dodano trzy dodatkowe aspekty takie jak: koalicja, klient oraz treść [58]. Dodane aspekty podkreślają poziomą integrację CPS jako koalicję różnych warstw i związanych z nimi informacjami (treściami), a także klienta jako najważniejszą stronę w procesie wytwarzania. Rysunek 2.1 przedstawia schemat architektury CPS 8C. Zagadnienia poruszane w niniejszej pracy dotyczą głównie warstwy 3 („Cyber"), w której realizowane są zadania sterowania fizycznymi procesami.

Powszechnie używanym sprzętem do implementacji części sterującej systemów cyber-fizycznych są między innymi sterowniki PLC (ang. Programmable Logic Controller) [59], [60], układy FPGA (ang. Field-Programmable Gate Array) jak również różnej klasy mikrokontrolery [61], [62], [63], [64]. W przypadku sterowników PLC (najczęściej wybieranych w branży przemysłowej) opracowano standard IEC 61131-3 opisujący graficzne oraz tekstowe języki programowania tych sterowników [65]. We wspomnianym standardzie przedstawiono pięć języków programowania, które można wykorzystać

w procesie implementacji części sterującej systemu cyber-fizycznego realizowanej w sterowniku PLC. Językami wymienionymi w standardzie są: Schemat Drabinkowy (ang. *Ladder diagram* – LD), Schemat Bloków Funkcyjnych (ang. *Function Block Diagram* – FBD), Sekwencyjna Karta Funkcji (ang. *Sequential Function Chart* – SFC), Tekst Strukturalny (ang. *Structured Text* – ST) oraz Lista Instrukcji (ang. *Instruction List* – IL). Ponadto język SFC formalnie oparty jest na sieciach Petriego [13], [66] co umożliwia adaptację opracowanych metod weryfikacji i analizy sieci Petriego. W przypadku Schematów Drabinkowych (LD) jest możliwość ich konwersji do języka SFC, co również umożliwia wykorzystanie metod analizy opracowanych dla modeli sieci Petriego.



Rysunek 2.1. Schemat architektury CPS 8C

2.2. Wprowadzenie do sieci Petriego

Sieci Petriego są graficznym i matematycznym narzędziem do modelowania różnego rodzaju procesów z wielu dziedzin nauki oraz przemysłu [67]. Twórcą teorii sieci Petriego jest Carl Adam Petri, który w roku 1962 opublikował swoją pracę doktorską pt. „Kommunikation mit Automaten" [68], będącą podstawą dalszych prac w tym obszarze. Dziś sieci Petriego stosowane są szeroko do modelowania procesów z różnych obszarów nauki oraz przemysłu (m.in. procesów produkcyjnych, biznesowych, biologicznych) [16], [69], [70], [71].

Bardzo intuicyjna forma graficzna połączona z technikami formalnej analizy oraz własnością intuicyjnego odzwierciedlania równoległości świetnie wpisuje się w proces modelowania systemów cyber-fizycznych, a zwłaszcza części sterującej tych systemów [72]. Sieć Petriego w formalnym zapisie jest reprezentowana przez skierowany graf dwudzielny, w którym występują dwa rodzaje wierzchołków: miejsca oraz tranzycje. Taki

podział typów wierzchołków umożliwia reprezentowanie stanów modelowanego systemu. Połączenia (łuki) mogą występować tylko pomiędzy różnymi typami wierzchołków co oznacza, że można łączyć tranzycję z miejscem oraz miejsce z tranzycją, ale miejsca z miejscem czy tranzycję z tranzycją już nie.

W dalszej części rozdziału zostaną wprowadzone niezbędne definicje związane z teorią grafów oraz sieciami Petriego, a także wybrane definicje dotyczące złożoności obliczeniowej i sposobów jej określania. Te ostatnie zostaną użyte w dalszej części rozprawy, w której zostaną opisane algorytmy analizy sieci Petriego, co umożliwi porównanie ich ze sobą.

2.2.1. Wybrane definicje dotyczące teorii grafów

W niniejszym podrozdziale zostaną wprowadzone niezbędne definicje oraz pojęcia z teorii grafów, które są później wykorzystywane w opisie oraz analizie sieci Petriego [67], [73].

Definicja 1. *Grafem (nieskierowanym)* nazywamy parę $G = (V, E)$ gdzie:

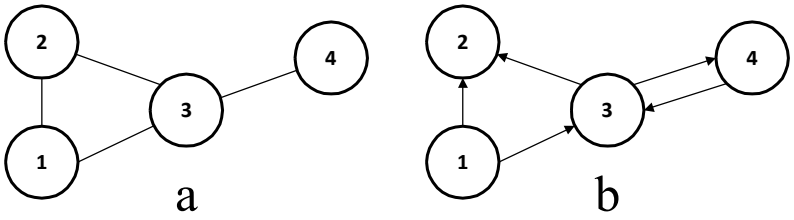
- $V = \{v_1, v_2, ..., v_n\}$ jest zbiorem wierzchołków,
- $E = \{e_1, e_2, ..., e_m\}$ jest zbiorem nieuporządkowanych par wierzchołków, zwanych krawędziami.

Definicja 2. *Grafem skierowanym* nazywamy parę $G = (V, E)$ gdzie:

- $V = \{v_1, v_2, ..., v_n\}$ jest zbiorem wierzchołków,
- $E = \{e_1, e_2, ..., e_m\}$ jest zbiorem uporządkowanych par wierzchołków, zwanych krawędziami.

Na rysunku 2.2 przedstawiano przykładowy graf nieskierowany (a) oraz skierowany (b). Graf nieskierowany z rysunku 2.2a składa się ze zbioru wierzchołków $V = \{1, 2, 3, 4\}$ oraz zbioru krawędzi $E = \{e_1, e_2, e_3, e_4\}$, w którym krawędziami są nieuporządkowane następujące pary wierzchołków: $e_1 = \{1, 2\}$, $e_2 = \{2, 3\}$, $e_3 = \{1, 3\}$, $e_4 = \{3, 4\}$. Należy pamiętać, że w przypadku grafu nieskierowanego w zbiorze krawędzi pary wierzchołków są nieuporządkowane co oznacza, że krawędź $e_1 = \{1, 2\}$ można zapisać również jako parę wierzchołków $e_1 = \{2, 1\}$. Oba zapisy są równoważne. Oczywiście opisana zależność dotyczy wszystkich krawędzi w zbiorze.

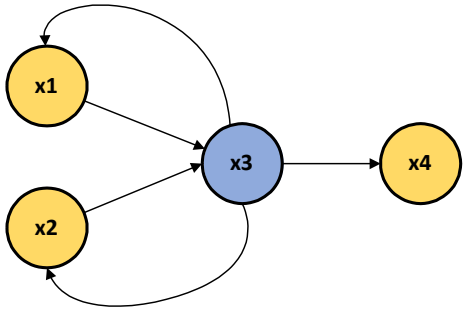
Na rysunku 2.2b graf skierowany składa się również ze zbioru wierzchołków $V = \{1, 2, 3, 4\}$, ale tym razem zbiór krawędzi $E = \{e_1, e_2, e_3, e_4, e_5\}$ tworzą uporządkowane pary wierzchołków: $e_1 = (1, 2)$, $e_2 = (1, 3)$, $e_3 = (3, 2)$, $e_4 = (3, 4)$, $e_5 = (4, 3)$.



Rysunek 2.2. Przykładowy graf (a) nieskierowany oraz (b) skierowany

Definicja 3. Graf skierowany $G = (V, E)$ nazywamy grafem skierowanym dwudzielnym, jeżeli zbiór wierzchołków V jest sumą dwóch rozłącznych zbiorów V_1 oraz V_2 i każda krawędź ze zbioru E łączy wierzchołki należące do dwóch różnych zbiorów.

Przykład grafu skierowanego dwudzielnego przedstawiono na rysunku 2.3. Graf składa się z dwóch zbiorów wierzchołków $V_1 = \{x_1, x_2, x_4\}$ oraz $V_2 = \{x_3\}$ (rozdzielne zbiory wierzchołków na grafie oznaczono odrębnymi kolorami).



Rysunek 2.3. Przykładowy graf skierowany dwudzielny

Definicja 4. *Zbiór następników* dla dowolnego wierzchołka $x \in V$ grafu skierowanego $G = (V, E)$ określamy jako $\boldsymbol{\mathcal{N}}(x) = \{y \in V: (x, y) \in E\}$.

Definicja 5. *Zbiór poprzedników* dla dowolnego wierzchołka $x \in V$ grafu skierowanego $G = (V, E)$ określamy jako $\boldsymbol{\mathcal{P}}(x) = \{y \in V: (y, x) \in E\}$.

Definicja 6. *Wierzchołkiem początkowym* nazywamy wierzchołek $x \in V$, dla którego zbiór poprzedników jest zbiorem pustym, $\boldsymbol{\mathcal{P}}(x) = \emptyset$.

Definicja 7. *Wierzchołkiem końcowym* nazywamy wierzchołek $x \in V$, dla którego zbiór następników jest zbiorem pustym, $\boldsymbol{\mathcal{N}}(x) = \emptyset$.

Definicja 8. *Wierzchołkiem izolowanym* nazywamy wierzchołek $x \in V$, dla którego zbiór następników oraz zbiór poprzedników jest pusty, $\boldsymbol{\mathcal{N}}(x) = \emptyset$ i $\boldsymbol{\mathcal{P}}(x) = \emptyset$.

Definicja 9. Niech graf $G = (V, E)$ będzie grafem skierowanym. *Drogą nieskierowaną* w grafie G nazywamy ciąg krawędzi $e_1, e_2,..., e_n \in E$ jeżeli istnieją wierzchołki $x_1, x_2,..., x_{(n+l)} \in V$ takie, że $\exists e_i \in E : (x_i, x_{(i+1)}) \vee (x_{(i+1)}, x_i)$ dla dowolnego $i = l, 2,..., n$. O drodze nieskierowanej mówimy wówczas, że prowadzi od wierzchołka x_1 do wierzchołka $x_{(n+1)}$.

Definicja 10. Niech graf $G = (V, E)$ będzie grafem skierowanym. *Drogą skierowaną* w grafie G nazywamy ciąg krawędzi $e_1, e_2,..., e_n \in E$ jeżeli istnieją wierzchołki $x_1, x_2, ..., x_{(n+l)} \in V$ takie, że $\exists e_i \in E : (x_i, x_{(i+1)})$ dla dowolnego $i = l, 2,..., n$. O drodze skierowanej mówimy wówczas, że prowadzi od wierzchołka x_1 do wierzchołka $x_{(n+1)}$.

Definicja 11. *Grafem spójnym* nazywamy graf $G = (V, E)$ jeżeli dla dwóch dowolnych wierzchołków $x \in V$ oraz $y \in V$ istnieje w grafie droga nieskierowana od x do y.

Definicja 12. *Grafem silnie spójnym* nazywamy graf $G = (V, E)$ jeżeli dla dwóch dowolnych wierzchołków $x \in V$ oraz $y \in V$ istnieje w grafie droga skierowana od x do y.

Definicja 13. *Składowa silnie spójna (komponent silnie spójny)* jest maksymalnym podgrafem grafu $G = (V, E)$, w którym istnieją drogi skierowane pomiędzy każdymi dwoma wierzchołkami.

2.2.2. Wybrane definicje oraz notacje dot. sieci Petriego

W tym podrozdziale zostaną wprowadzone niezbędne definicje oraz pojęcia, które będą wykorzystywane w opisie metod analizy oraz weryfikacji właściwości (w szczególności żywotności) sieci Petriego modelującej część sterującą systemu cyber-fizycznego [27], [39], [62], [67], [74], [75], [76], [77].

Definicja 14. *(Znakowaną) siecią Petriego* nazywamy uporządkowaną czwórkę $N = (P, T, F, M_0)$ gdzie:

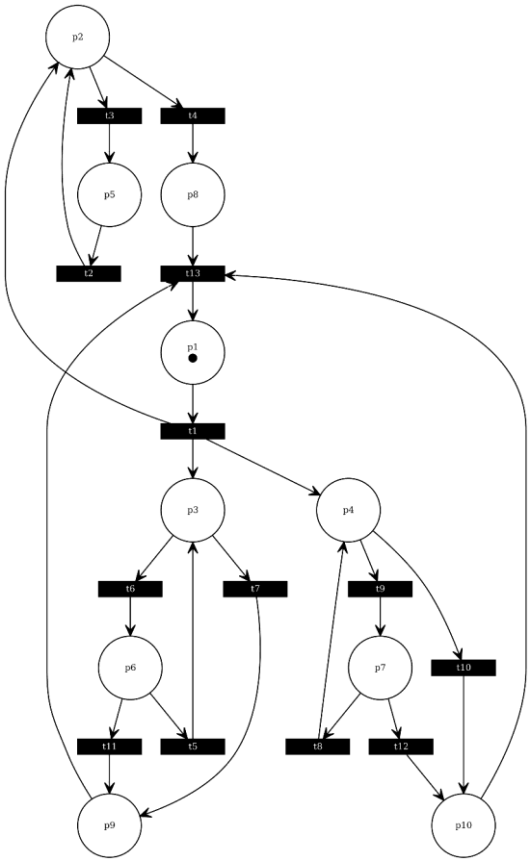
- P jest skończonym, niepustym zbiorem miejsc,
- T jest skończonym, niepustym zbiorem tranzycji takim, że $P \cap T = \emptyset$,
- $F \subseteq (P \times T) \cup (T \times P)$ jest skończonym, niepustym zbiorem krawędzi,
- M_0: $P \rightarrow Z_+$ jest funkcją określoną na zbiorze miejsc, zwaną znakowaniem początkowym sieci N.

Dla sieci Petriego przyjmuje się następującą interpretację. Sieć przedstawia się jako skierowany graf dwudzielny, którego wierzchołkami są elementy zbiorów P oraz T, ,krawędziami zaś pary relacji F. Miejsca sieci są reprezentowane graficznie za pomocą elips

lub okręgów, tranzycje zaś za pomocą prostokątów. Relacja przepływu jest reprezentowana za pomocą strzałek łączących odpowiednio elipsy z prostokątami i prostokąty z elipsami.

Znakowanie początkowe jest funkcją, która każdemu miejscu sieci przyporządkowuje nieujemną liczbę całkowitą, interpretowaną jako liczba znaczników (żetonów, tokenów) umieszczonych wstępnie w tym miejscu. Znaczniki są reprezentowane graficznie za pomocą kropek umieszczanych wewnątrz elips symbolizujących miejsca lub gdy liczba znaczników jest duża, za pomocą etykiet zawierających informację o liczbie znaczników.

Na rysunku 2.4 przedstawiono znakowaną sieć Petriego, która jest formalnym modelem systemu produkcyjnego wyposażonego w trzy strefy bezpieczeństwa oraz komputerowy system wizyjny (sieć jest dostępna w systemie Hippo pod nazwą *Manufacturing system with three zones and computer vision system*). Model ten odwzorowuje system opisany w rozdziale pierwszym. W znakowaniu początkowym znacznik znajduje się w miejscu *p1*. Zbiór miejsc P składa się z 10 miejsc {*p1*, *p2*, *p3*, *p4, p5, p6, p7, p8, p9, p10*}, a zbiór tranzycji T zawiera 13 tranzycji {*t1*, *t2*, *t3*, *t4*, *t5*, *t6, t7, t8, t9, t10, t11, t12, t13*}.



Rysunek 2.4. Sieć Petriego reprezentująca model systemu produkcyjnego wyposażonego w trzy strefy bezpieczeństwa oraz komputerowy system wizyjny ze znakowaniem początkowym

Definicja 15. Tranzycję $t \in T$ nazywamy *tranzycją wejściową* do miejsca $p \in P$ jeżeli $(t, p) \in F$.

Definicja 16. Zbiór tranzycji wejściowych do miejsca $p \in P$ definiuje się jako $\bullet p = \{t \in T : (t, p) \in F\}$.

Definicja 17. Tranzycję $t \in T$ nazywany *tranzycją wyjściową* z miejsca $p \in P$ jeżeli $(p, t) \in F$.

Definicja 18. Zbiór tranzycji wyjściowych z miejsca $p \in P$ definiuje się jako $p\bullet = \{t \in T : (p, t) \in F\}$.

Definicja 19. Miejsce $p \in P$ nazywany *miejscem wejściowym* do *tranzycji* $t \in T$ jeżeli $(p, t) \in F$.

Definicja 20. Zbiór miejsc wejściowych do tranzycji $t \in T$ definiuje się jako $\bullet t = \{p \in P : (p, t) \in F\}$.

Definicja 21. Miejsce $p \in P$ nazywany *miejscem wyjściowym* z tranzycji $t \in T$ jeżeli $(t, p) \in F$.

Definicja 22. Zbiór miejsc wyjściowych z tranzycji $t \in T$ definiuje się jako $t\bullet = \{p \in P : (t, p) \in F\}$.

Definicja 23. Tranzycja $t \in T$ jest *aktywna* w znakowaniu M, jeżeli w każdym miejscu wejściowym tranzycji w tym znakowaniu znajduje się co najmniej jeden znacznik (żeton, token).

Definicja 24. *Znakowaniem sieci Petriego* nazywamy dowolną funkcję $M: P \rightarrow Z_+$. Znakowanie sieci zmienia się w momencie odpalenia tranzycji. Odpalenie tranzycji powoduje usunięcie jednego znacznika z każdego miejsca wejściowego odpalonej tranzycji oraz dodanie jednego znacznika do każdego miejsca wyjściowego odpalonej tranzycji. Tranzycja może zostać odpalona wtedy i tylko wtedy, gdy jest *aktywna*.

Definicja 25. *Znakowaniem osiągalnym* ze znakowania M nazywamy dowolne znakowanie M' sieci, które można osiągnąć ze znakowania M w wyniku wykonania skończonego ciągu przejść $\sigma = t_1, t_2, \ldots, t_n \in T$. gdzie każde t_i (dla $i = 1, 2, \ldots, n$) jest przejściem w sieci, a każde przejście w ciągu σ jest wykonalne w momencie jego uruchomienia (czyli w znakowaniu powstałym po wcześniejszych przejściach).

Definicja 26. Zbiór wszystkich znakowań osiągalnych ze znakowania M oznacza się $R(M)$. Zbiór znakowań osiągalnych ze znakowania początkowego M_0 oznacza się symbolem $R(M_0)$. Z każdym znakowaniem osiągalnym ze znakowania M jest związany ciąg przejść, których wykonanie prowadzi od znakowania M do M'. Zbiór wszystkich ciągów przejść, które można wykonać rozpoczynając od znakowania M oznacza się $L(M)$. Zbiór $L(M)$ może zawierać zarówno skończone, jak i nieskończone ciągi przejść.

Definicja 27. Miejsce $p \in P$ nazywamy *znakowanym*, jeżeli istnieje znakowanie $M \in R(M_0)$ takie, że $M(p) > 0$.

Definicja 28. Miejsce $p \in P$ nazywamy miejscem *ograniczonym*, jeżeli spełniony jest warunek taki, że $\exists k \in \mathbb{N}\ \forall M \in R(M_0)$: $M(p) \leq k$. Miejsce, które spełnia przytoczony warunek można również nazwać *k-ograniczonym*, gdzie k jest *górnym ograniczeniem* miejsca p. Miejsce p nazywamy *bezpiecznym*, gdy jest *1-ograniczone*. Sieć Petriego N nazywamy *k-ograniczoną* (lub po prostu *ograniczoną*), kiedy wszystkie jej miejsca są *k-ograniczone*. Sieć Petriego N nazywamy *bezpieczną*, kiedy wszystkie jej miejsca są *bezpieczne* (*1-ograniczone*).

Definicja 29. Znakowanie początkowe M_0 nazywamy *odtwarzalnym*, jeżeli istnieje znakowanie $M \in R(M_0)$ różne od znakowania M_0, z którego znakowanie początkowe jest ponownie osiągalne. Sieć Petriego N nazywamy *odtwarzalną*, jeżeli znakowanie początkowe jest odtwarzalne. Sieć Petriego N nazywamy *odwracalną,* jeżeli znakowanie początkowe jest osiągalne z każdego znakowania $M \in R(M_0)$.

Definicja 30. Niech tranzycja $t \in T$ będzie tranzycją w sieci $N = (P, T, F, M_0)$,

- tranzycję t nazywamy *L0-żywą* (martwą), jeżeli tranzycja t nie występuje w żadnym ciągu tranzycji należącym do zbioru $L(M_0)$,
- tranzycję t nazywamy *L1-żywą* (potencjalnie wykonalną), jeżeli istnieje ciąg tranzycji należących do zbioru $L(M_0)$, w którym tranzycja t występuje co najmniej jeden raz,
- tranzycję t nazywamy *L2-żywą*, jeżeli dla dowolnej liczby naturalnej k ($k > 1$) istnieje ciąg tranzycji należący do zbioru $L(M_0)$, w którym tranzycja t występuje co najmniej k razy,
- tranzycję t nazywamy *L3-żywą*, jeżeli istnieje ciąg tranzycji należący do zbioru $L(M_0)$, w którym tranzycja t występuje nieskończenie wiele razy,

- tranzycję t nazywamy *L4-żywą* (żywą), jeżeli tranzycja t jest potencjalnie wykonalna dla każdego znakowania $M \in R(M_0)$.

Sieć Petriego N nazywamy *Lk-żywą*, jeżeli każda tranzycja tej sieci jest Lk-żywa dla $k = 0, 1, 2, 3, 4$. Sieć Petriego N nazywamy *dokładnie Lk-żywą*, jeżeli jest ona *Lk-żywa*, ale nie jest *L(k+1)*-żywa. Sieć Petriego N nazywamy *strukturalnie żywą*, jeżeli istnieje znakowanie początkowe M_0, dla którego ta sieć jest żywa. Sieć L4-żywą będziemy w niniejszej rozprawie nazywać po prostu *siecią żywą*.

Definicja 31. Grafem osiągalności sieci Petriego N nazywamy graf $G = (V, E)$, gdzie $V=R(M_0)$ jest zbiorem węzłów grafu, $E = \{(M, t, M'):M, M' \in R(M_0) \wedge M \xrightarrow{t} M'\}$ jest zbiorem łuków etykietowanych nazwami tranzycji, których wykonanie powoduje zmianę znakowania z M na M'.

Definicja **32**. *Macierzą incydencji* sieci $N = (P, T, F, M_0)$ nazywamy macierz $A_{|T|\times|P|}$ z $|T|$ wierszami oraz $|P|$ kolumnami liczb całkowitych, których wartości opisane są:

$$a_{ij} = \begin{cases} -1, (p_j, t_i) \in F, \\ 1, (t_i, p_j) \in F, \\ 0, \text{w przeciwnym razie} \end{cases}$$

Definicja 33. Sieć Petriego N nazywamy *maszyną stanową* SM, jeżeli każda jej tranzycja ma dokładnie jedno miejsce wejściowe oraz dokładnie jedno miejsce wyjściowe.

Definicja 34. Sieć Petriego N nazywamy *grafem znakowań* MG, jeżeli każde jej miejsce ma dokładnie jedną tranzycję wejściową oraz dokładnie jedną tranzycję wyjściową.

Definicja 35. Sieć Petriego N nazywamy *siecią swobodnego wyboru* FC, jeżeli dla dowolnych dwóch miejsc w sieci N, które mają wspólną tranzycję wyjściową ta tranzycja jest ich jedyną tranzycją wyjściową.

Definicja 36. Sieci Petriego N nazywamy *rozszerzoną siecią swobodnego wyboru* EFC, jeżeli dla dowolnych dwóch miejsc w sieci N, które mają wspólną tranzycję wyjściową, mają równe zbiory tranzycji wyjściowych.

Definicja 37. Sieci Petriego N nazywamy *asymetryczną siecią swobodnego wyboru* AC, jeżeli dla dowolnych dwóch miejsc w sieci N, które mają wspólną tranzycję wyjściową, zbiór tranzycji wyjściowych jednego z tych miejsc zawiera się w zbiorze tranzycji wyjściowych drugiego miejsca.

2.3. Złożoność algorytmu

W tym podrozdziale zostaną wprowadzone zagadnienia związane ze złożonością algorytmów oraz samymi algorytmami. Analizę właściwości sieci Petriego można interpretować jako problem obliczeniowy do rozwiązania. Algorytm jest zapisem kroków, za pomocą których można rozwiązać problem obliczeniowy. Algorytmy przeważnie można podzielić na trzy części. Pierwsza część to dane wejściowe, które w zależności od algorytmu muszą zostać przekonwertowane. Drugą częścią jest realizacja kroków, które umożliwią rozwiązanie problemu, a trzecią częścią są dane wyjściowe, które podobnie jak dane wejściowe mogą wymagać przekształcenia w celu zrozumiałego dla człowieka wyniku. W kontekście realizowanej rozprawy danymi wejściowymi będą sieci Petriego, które modelują część sterującą systemu CPS. Danymi wyjściowymi będą informacje o wyniku weryfikacji bądź analizy np. sieć wejściowa jest żywa, jest bezpieczna itp.

Dla każdego algorytmu można określić tak zwaną złożoność obliczeniową, która umożliwia określenie jego wydajności. Po wyznaczeniu złożoności jest możliwość porównania go z innymi algorytmami (które rozwiązują ten sam problem obliczeniowy) w celu wybrania najlepszego rozwiązania. W celu zdefiniowania złożoności obliczeniowej wprowadzono w pracy następujące pojęcia i definicje [78]:

Definicja 38. *Złożoność czasowa algorytmu* to funkcja $f(n)$, która określa maksymalną liczbę iteracji algorytmu dla dowolnego wejścia o długości n.

Definicja 39. *Górną granicą dla funkcji* $f(n)$ jest funkcja $g(n)$ taka, że $f(n) = O(g(n))$.

Definicja 40. Algorytm jest ograniczony wielomianem o wielkości wejścia n, jeżeli liczbę iteracji w algorytmie można oszacować za pomocą funkcji $f(n) = O(n^c)$, gdzie $c > 0$.

Definicja 41. Algorytm jest ograniczony wykładniczym rozmiarem danych wejściowych n, jeżeli liczbę iteracji w algorytmie można oszacować za pomocą funkcji $f(n) = O(c^n)$, gdzie $c > 1$.

Definicja 42. *Złożoność wielomianowa* (czas wielomianowy) algorytmu wskazuje, że całkowity czas działania algorytmu w celu wygenerowania wszystkich danych wyjściowych jest ograniczony wielomianem wielkości danych wejściowych.

Definicja 43. *Złożoność wykładnicza* (czas wykładniczy) algorytmu oznacza, że całkowity czas działania algorytmu w celu wygenerowania wszystkich danych wyjściowych jest ograniczony wykładniczym rozmiarem danych wejściowych.

2.4. Przegląd wybranych metod analizy Sieci Petriego

W procesie modelowania części sterującej systemów cyber-fizycznych z wykorzystaniem sieci Petriego bardzo ważnym etapem jest analiza modelu przed jego fizyczną implementacją. W zależności od tego, jakie były założenia początkowe projektanta dot. modelowanej części sterującej systemu CPS, może być konieczne potwierdzenie (za pomocą specjalnych metod analizy) wybranych właściwości modelu wyrażonego siecią Petriego. Takimi właściwościami mogą być między innymi bezpieczeństwo, ograniczoność, odwracalność, żywotność. W niniejszym rozdziale skupiono się na najistotniejszych właściwościach związanych z *żywotnością* systemu opisanego siecią Petriego. Przy czym warto w tym miejscu zauważyć, że niektóre właściwości są ze sobą powiązane i analiza jednej z nich może wymagać weryfikacji innej. Na przykład analiza żywotności z wykorzystaniem grafu osiągalności wymaga najpierw weryfikacji ograniczoności danej sieci, jeszcze przed rozpoczęciem docelowej procedury weryfikacji żywotności (ten aspekt zostanie opisany szerzej w rozdziale 4). Dlatego też w pierwszej kolejności opisano ograniczoność sieci (oraz pokrewną właściwość jaką jest bezpieczeństwo sieci), a następnie skoncentrowano się na aspekcie żywotności.

2.4.1. Ograniczoność i bezpieczeństwo

Ograniczoność i bezpieczeństwo są właściwościami, które bardzo często wymagane są przez projektantów części sterującej systemów cyber-fizycznych [79], [80]. W rozdziale 2.2.2 (definicja 28) przedstawiono w sposób formalny czym jest ograniczoność oraz bezpieczeństwo. Używając mniej formalnej definicji ograniczoność można opisać w następujący w sposób: w trakcie działania sieci, w każdym jej miejscu liczba tokenów (żetonów) ma swój górny limit, który nie zostanie przekroczony (liczba żetonów nie będzie dążyć do nieskończoności). W przypadku bezpieczeństwa warunek jest bardziej restrykcyjny niż przy ograniczoności. Aby sieć była bezpieczna to w trakcie działania sieci, w każdym jej miejscu liczba tokenów nie może przekroczyć jednego. W tym rozdziale zostaną opisane najczęściej wykorzystywane algorytmy do weryfikacji ograniczoności oraz bezpieczeństwa.

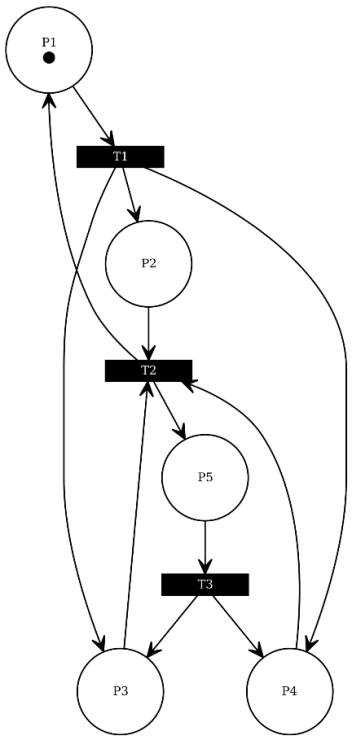
Na rysunku 2.5 przedstawiono przykładową sieć Petriego, która nie jest ograniczona[2]. Jest to sieć o nazwie *pn_silva_05c,* która jest częścią bazy testowej *Hippo* (baza *Hippo* została opisana szczegółowo w rozdziale 5.1). Można zauważyć, że wybrana sieć nie jest duża, ponieważ składa się z tylko pięciu miejsc oraz trzech tranzycji.

[2] w pracy przyjęto, że sieć, która nie jest ograniczona nazywana będzie także siecią *nieograniczoną*

W przedstawionej sieci początkowo znakowanym miejscem jest P1. W trakcie działania sieci w miejscach P3 oraz P4 będą kumulować się tokeny.

W literaturze można znaleźć kilka podejść do weryfikacji ograniczoności. Pierwszą techniką jest wykorzystanie tzw. *grafu pokrycia* [27], [36], [39]. W takim grafie specjalnym symbolem oznaczane są miejsca, w których gromadzone są tokeny. Podejście to jest bardzo skuteczne, ale obarczone dość dużym ograniczeniem, którym jest wykładnicza złożoność obliczeniowa. W procesie wyznaczania grafu pokrycia może również wystąpić problem znany pod nazwą *eksplozji stanów*, który oznacza bardzo szybki przyrost nowych stanów podczas analizowania sieci Petriego. Taki przyrost stanów może uniemożliwić wyznaczenie grafu pokrycia. Zjawisko eksplozji stanów nie jest bezpośrednio skorelowane z liczbą miejsc oraz tranzycji, lecz raczej sposobie oraz liczbą ich połączeń.

Wspomniany graf porycia (a dokładniej algorytm do generowania grafu pokrycia) można również wykorzystać do weryfikacji bezpieczeństwa. W przypadku sieci, która jest ograniczona, wynikiem algorytmu wyznaczającego graf pokrycia będzie graf osiągalności (w takim grafie nie występuje symbol oznaczający miejsce nieograniczone). W przypadku analizy bezpieczeństwa, w grafie osiągalności poszukiwane są stany, w których miejsca posiadają więcej niż jeden token, co oznacza, że badana sieć nie jest bezpieczna, w przeciwnym przypadku sieć jest bezpieczna (szczegółowy opis algorytmu do generowania grafu pokrycia znajduje się w rozdziale 3.1 poświęconym metodzie referencyjnej do weryfikacji żywotności).



Rysunek 2.5. Przykład sieci (*pn_silva_05c*), która nie jest ograniczona

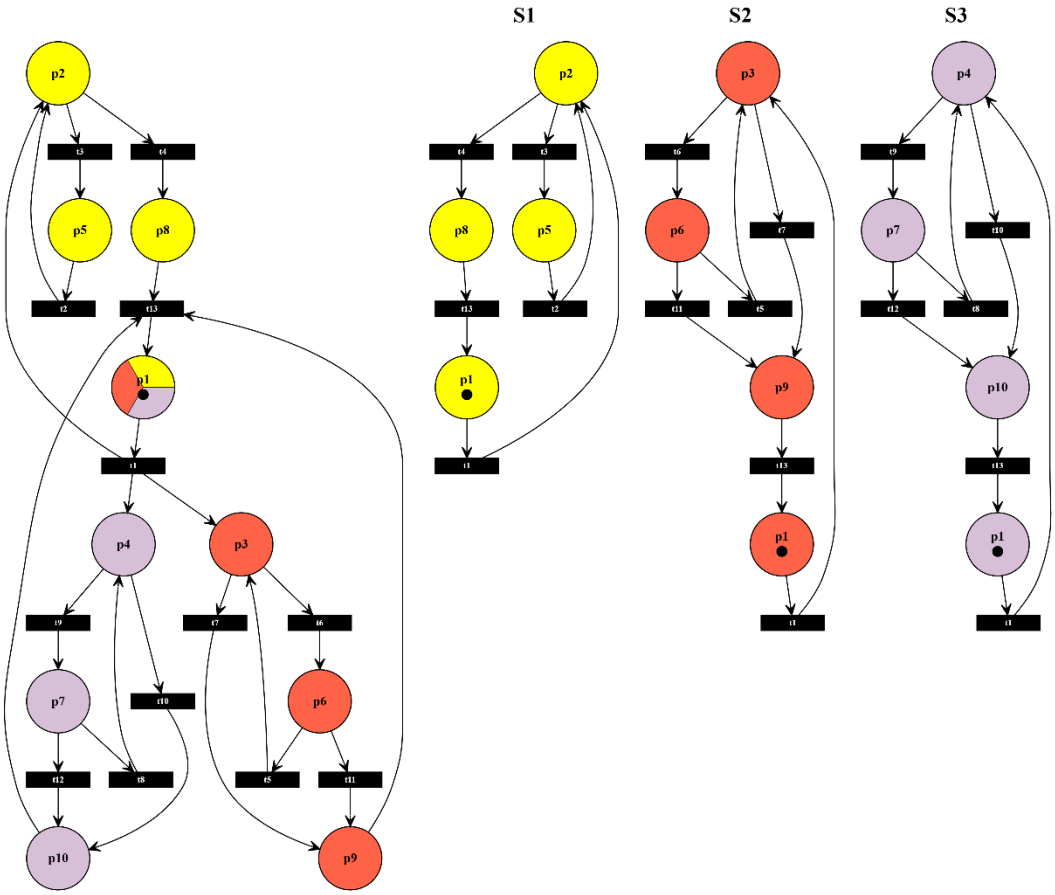
Drugą bardzo popularną techniką do weryfikacji ograniczoności jest wykorzystanie tzw. inwariantów (niezmienników) miejsc. Sam inwariant można opisać jako zbiór miejsc, w którym łączna suma znaczników się nie zmienia. Główną ideą tej techniki jest sprawdzenie czy sieć jest pokryta przez niezmienniki miejsc (pokrycie oznacza, że każde miejsce sieci wystąpiło przynajmniej raz w którymś z wyznaczonych niezmienników miejsc). Jeżeli takie pokrycie występuje w sieci oznacza to, że suma ważona znaczników dla każdego znakowania $\mathbf{M}$ jest stała. Wynika z tego, że nie występuje w takiej sieci miejsce, w którym tokeny mogą gromadzić się w nieskończoność.

Tak jak w przypadku badania ograniczoności sieci, także do weryfikacji bezpieczeństwa można wykorzystać niezmienniki miejsc, ale w przypadku bezpieczeństwa wprowadzona jest dodatkowa restrykcja, która mówi, że pokrycie musi nastąpić przez niezmienniki miejsc, w których jest jeden token. Tego typu niezmiennik miejsc nazywany jest maszyną stanową *(ang. state-machine component*, SMC) [21], [27], [62].

W literaturze opisano kilka różnych technik do wyznaczania inwariantów [81], [82], [83], ale najbardziej popularną jest technika oparta na tzw. algorytmie *Martinez-Silva* (nazwa przylgnęła od twórców metody) [35]. W literaturze można też odnaleźć algorytmy, które opierają się na algorytmie *Martinez-Silva*, ale wprowadzają do niego usprawnienia znacznie poprawiające jego efektywność działania [32], [33], [39], [84]. Na rysunku 2.6 przedstawiono sieć o nazwie *Manufacturing system with three zones and computer vision system*, na której różnymi kolorami oznaczono inwarianty, które pokrywają całą sieć. Z kolei z prawej strony rysunku poglądowo zamieszczono inwarianty, które mogą zostać wyznaczone w omawianej sieci. W przedstawionej sieci można zauważyć, że każdy z wyznaczonych inwariantów zawiera token (wyznaczone inwarianty są maszynami stanowymi) co oznacza, że sieć jest nie tylko ograniczona, ale również bezpieczna.

Bardzo ciekawą alternatywą do weryfikacji ograniczoności jest wykorzystanie tzw. *zredukowanej macierzy schodkowej* [85]. To rozwiązanie podobne jest do podejścia wykorzystującego inwarianty miejsc (pokrycie przez niezmienniki miejsc), ale nie wymaga wyznaczania tych niezmienników, co stanowczo przyspiesza działanie tej metody [86]. Główna idea tego rozwiązania opiera się na przekształceniu macierzy incydencji na zredukowaną macierz schodkową (przy wykorzystaniu elementarnych operacji macierzowych). Po otrzymaniu zredukowanej macierzy algorytm poszukuje nieograniczonych miejsc w sieci. W szczególności, metoda wyszukuje wiersze zawierające wyłącznie wartości dodatnie, które nie są „kasowane" przez żaden inny wiersz. Istnienie

takich wartości przesądza, że miejsca odpowiadające tym wpisom nie mogą tworzyć właściwego inwariantu, a co za tym idzie, są potencjalnie nieograniczone [87].



Rysunek 2.6. Wizualizacja pokrycia inwariantami dla sieci *Manufacturing system with three zones and computer vision system*

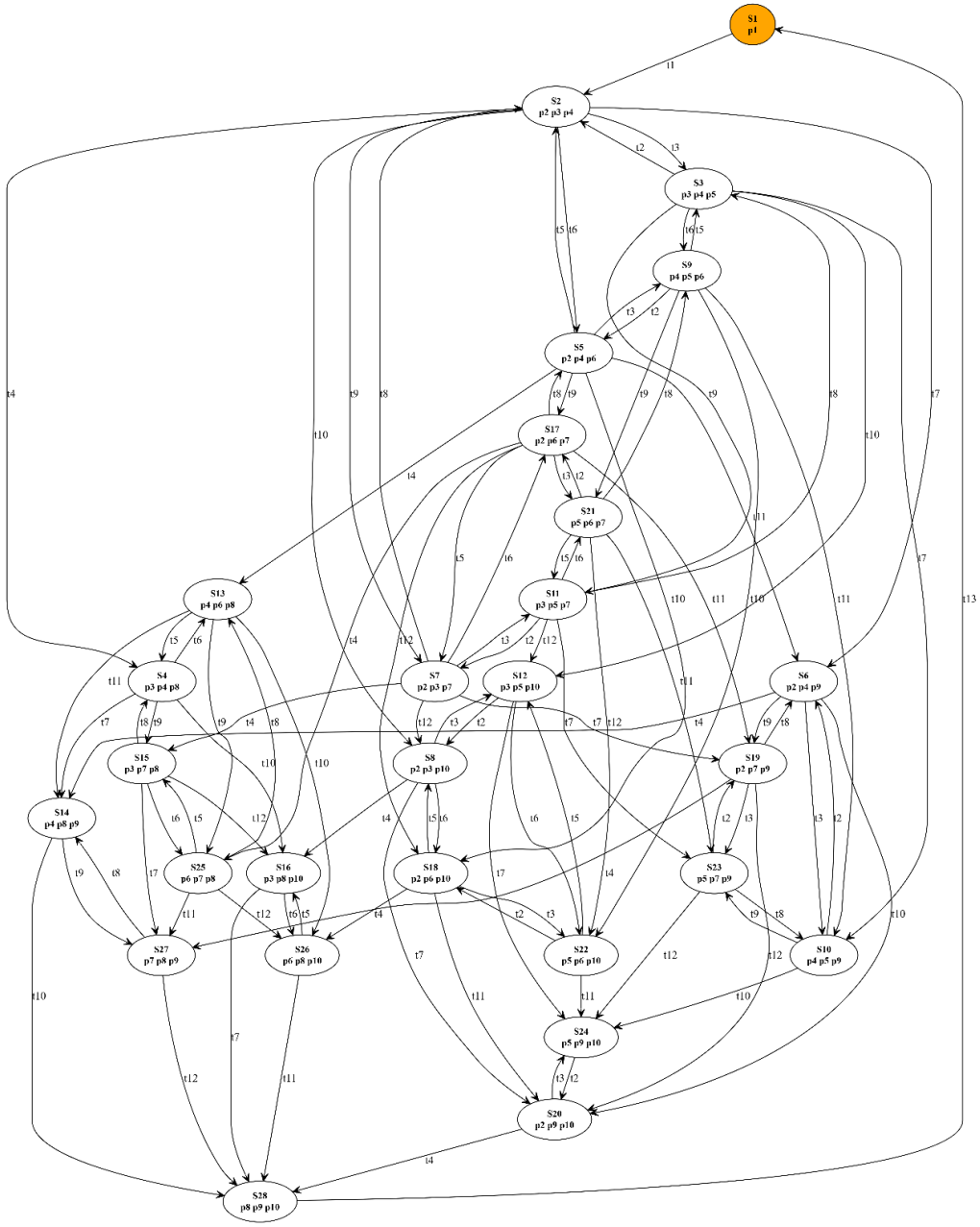
2.4.2. Żywotność

Żywotność jest bardzo ciekawą i jednocześnie istotną właściwością sieci Petriego. Przede wszystkim sieć, która jest żywa (bardziej formalnie: L4-żywa), nie posiada zakleszczeń (ang. *deadlocks*). Co więcej, w tego typu sieci każda tranzycja może zostać odpalona (np. w wyniku odpalenia innych tranzycji). Innymi słowy, nie istnieje tranzycja, która nigdy nie mogłaby zostać odpalona. Z tego też względu, żywotność jest jedną z kluczowych własności badanych już na wczesnym etapie modelowania części sterującej systemu cyber-fizycznego [29], [31], [32]. Z drugiej strony, przeprowadzony przegląd literatury jednoznacznie wskazuje, że analiza żywotności jest zdecydowanie bardziej utrudniona (zwłaszcza pod kątem czasu obliczeń), niż analiza innych własności systemu (np. ograniczoności czy bezpieczeństwa). Głównym problemem jest wykładnicza złożoność obliczeniowa algorytmów umożliwiających weryfikację żywotności sieci Petriego (w ogólnym przypadku). Ograniczeniem są również dostępne techniki badania

żywotności. Przykładowo, w odróżnieniu od np. ograniczoności sieci, która może zostać zweryfikowana z zastosowaniem różnych podejść (graf osiągalności, algebra liniowa), analiza żywotności najczęściej bazuje na konstrukcji grafu osiągalności [27], [31], [67]. Wprawdzie w literaturze przedmiotu znaleźć można inne podejścia do analizy żywotności modeli opisanych sieciami Petriego [88], [89], [90], [91], to jednak mają one istotne ograniczenia, najczęściej zawężając badany system do wybranych klas sieci Petriego. Poniżej krótko scharakteryzowano najczęściej wykorzystywane techniki badania żywotności sieci, z uwzględnieniem ich zalet oraz wad.

Analiza grafu osiągalności jest zdecydowanie najczęściej stosowaną metodą weryfikacji żywotności systemu opisanego siecią Petriego [27], [31], [32], [34], [74], [92], [93], [94]. Ogólnie mówiąc, podejście to polega na wyznaczeniu grafu osiągalności (w takim grafie znajdują się wszystkie osiągalne stany w sieci ze znakowania początkowego) oraz weryfikacji czy w wyznaczonym grafie osiągalności można wyznaczyć taką drogę z wybranego wierzchołka (stanu) tego grafu do tego samego wierzchołka (stanu), aby w jej składzie znajdowały się wszystkie tranzycje. Na rysunku 2.7 przedstawiono poglądowy graf osiągalności stanów dla sieci *Manufacturing system with three zones and computer vision system*. W przedstawionym grafie osiągalności stanów, stan *S1* jest oznaczony kolorem, ponieważ jest to stan początkowy (wynikający ze znakowania początkowego sieci). Pod każda nazwą stanu znajdują się nazwy miejsc, które są znakowane w tym stanie.

Szczegółowa analiza grafu osiągalności dla tej sieci byłaby mało czytelna, ponieważ zawiera on znaczną liczbę wierzchołków (stanów) oraz łuków. Z tego powodu zdecydowano, aby w dalszej części pracy (w rozdziale poświęconym metodzie referencyjnej) przeprowadzić analizę innej sieci o nazwie *3carros* (także dostępnej w systemie Hippo). Jej graf osiągalności jest bardziej przejrzysty, co ułatwia jego wizualną interpretację i szczegółowy opis, a tym samym sprzyja bardziej czytelnemu przedstawieniu procesu analizy. W tym samym rozdziale szczegółowo opisano również aspekty analizy żywotności sieci Petriego w oparciu o ten graf osiągalności.

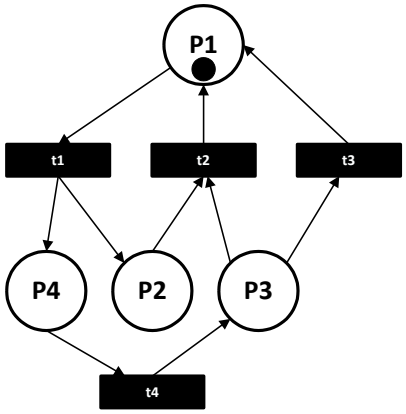


Rysunek 2.7. Graf osiągalności dla sieci *Manufacturing system with three zones and computer vision system*

Wspomniana metoda weryfikacji żywotności wykorzystująca graf osiągalności została wybrana jako metoda referencyjna, do której została porównana proponowana w pracy metoda (szczegółowy opis metody referencyjnej zaprezentowano w rozdziale 4, z kolei proponowaną metodę opisano w rozdziale 5). Zaletą metody bazującej na grafie osiągalności jest jej uniwersalność. Tego typu podejście umożliwia analizę żywotności dowolnej sieci Petriego (tzn. bez zawężenia do poszczególnych klas). Z drugiej strony, zwłaszcza w klasycznym podejściu, metoda charakteryzuje się wykładniczą złożonością obliczeniową. Oznacza to, że istnieje ryzyko, iż poszukiwane rozwiązanie nie zostanie znalezione w założonym (satysfakcjonującym) czasie. Ponadto, metoda wymaga, aby

badana sieć była ograniczona (ta właściwość jest badana w trakcie analizy żywotności). Wynika to z budowy grafu osiągalności, który do poprawnej analizy musi mieć ograniczoną liczbę stanów osiągalnych (więcej informacji o tym aspekcie znajduje się w rozdziale 4). Niemniej jednak warto w tym miejscu dodać, że ograniczoność sieci jest własnością, która najczęściej również musi zostać spełniona w modelowanym systemie, wobec czego ten wymóg można poniekąd traktować nie jako wadę, lecz wręcz zaletę metody.

Inną metodą weryfikacji żywotności, którą można znaleźć w literaturze jest analiza występowania w sieci pułapek oraz zatrzasków [91], [95], [96], [97], [98], [99]. Zatrzaskiem (ang. *siphon*) nazywamy zbiór miejsc nieposiadających tokenów (puste miejsca) w pewnym stanie S i pozostających pustymi w każdym stanie osiągalnym ze stanu S. W przypadku pułapki (ang. *trap*) można powiedzieć, że sytuacja jest odwrotna, ponieważ jest to zbiór miejsc, który zawiera tokeny w pewnym stanie S i w każdym stanie osiągalnym z tego stanu S ten zbiór miejsc będzie zawierać tokeny. Na rysunku 2.8 przedstawiono sieć zaczerpniętą z literatury [27], która jest *asymetryczną siecią swobodnego wyboru* (AC), dla której wyznaczono pułapki oraz zatrzaski.



Rysunek 2.8. Przykładowa sieć klasy AC

W sieci z rysunku 2.8 znajdują się dwa zatrzaski oraz dwie pułapki. Zatrzask pierwszy składa się z miejsc P1, P3, P4, zatrzask drugi składa się z miejsc P1, P2, P3, P4. W skład pułapki pierwszej wchodzą miejsca P1, P3, P4, a w pułapce drugiej są miejsca P1 oraz P2.

Główną ideą tej metody jest weryfikacja, czy w sieci każdy zatrzask zawiera w sobie znakowaną pułapkę. Należy w tym miejscu dodać, że wspominana technika weryfikacji żywotności ogranicza się tylko do wybranych klas sieci Petriego takich jak: sieci swobodnego wyboru (FC), rozszerzone sieci swobodnego wyboru (EFC) oraz asymetryczne sieci swobodnego wyboru (AC). Sieć na rysunku 2.8 spełnia warunki, które

wymagane są, aby ta sieć była żywa. Zatrzask pierwszy oraz drugi zawierają znakowane pułapki.

Dla innych klas sieci Petriego występują również techniki weryfikacji żywotności, np. dla sieci z klasy *maszyny stanowe* (*SM*) aby zweryfikować żywotność zależy sprawdzić, czy sieć jest silnie spójna oraz czy w znakowaniu początkowym sieci M_0 znajduje się przynajmniej jeden token [27], [67]. Można zauważyć, że jedynie technikę opartą o graf osiągalności można zastosować dla dowolnych sieci (bez analizy do jakiej klasy należy ta sieć) i dlatego właśnie wybrano ją jako metodę referencyjną. Proponowane w pracy autorskie rozwiązanie także można wykorzystać dla dowolnej klasy sieci Petriego.

2.5. Podsumowanie

W rozdziale krótko scharakteryzowano czym są systemy cyber-fizyczne oraz przedstawiono najbardziej popularne architektury systemów CPS wykorzystywane do projektowania oraz implementacji systemów cyber-fizycznych. Wskazano także warstwę CPS, w której mieszczą się zagadnienia poruszane w pracy. Wspomniano również o platformach realizacyjnych, które można wykorzystać do implementacji części sterującej systemu cyber-fizycznego, a także wskazano standard IEC 61131-3 definiujący języki programowania wykorzystywane do programowania sterowników PLC. Dwa z zawartych w nim języków są powiązanie z sieciami Petriego, co potencjalnie umożliwia weryfikację poprawności modeli sterowników PLC z wykorzystaniem metod analizy opracowanych dla sieci Petriego.

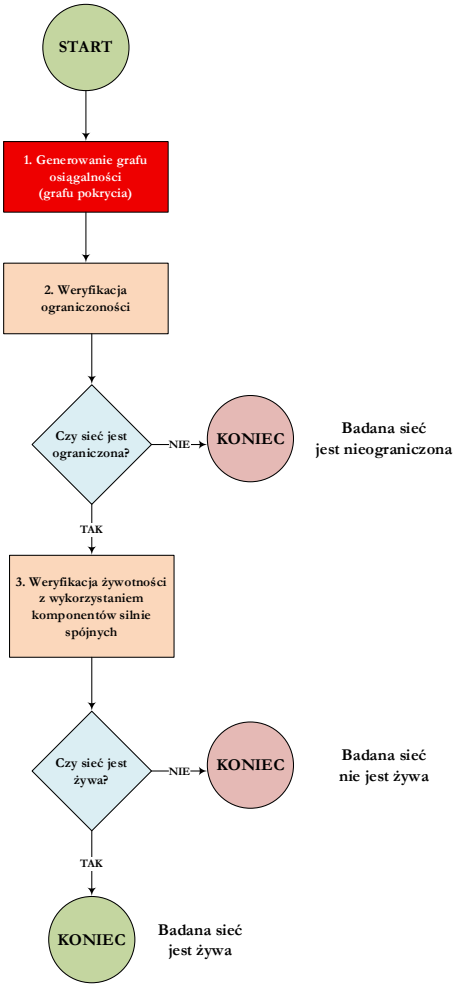
W rozdziale wprowadzono również zagadnienia teoretyczne związane z sieciami Petriego oraz złożonością obliczeniową algorytmów. Wprowadzono niezbędne definicje z teorii grafów, na których formalnie oparte są sieci Petriego. Przedstawiono także wybrane definicje oraz zagadnienia z teorii sieci Petriego. Wprowadzenie do pracy zagadnień związanych ze złożonością obliczeniową umożliwiło obiektywne porównanie algorytmów, które zostaną zaprezentowane w dalszej części pracy.

Ponadto, w rozdziale przedstawiono najważniejsze właściwości sieci Petriego: ograniczoność, bezpieczeństwo oraz żywotność. Zaprezentowano przykładowe sieci Petriego oraz na ich podstawie omówiono sposób weryfikacji ograniczoności oraz bezpieczeństwa sieci. Omówiono również inne techniki weryfikacji wybranych właściwości. W przypadku właściwości żywotności przedstawiono najbardziej znane metody oraz wskazano jedną z nich jako metodę referencyjną, do której będzie porównana proponowana autorska metoda.

3. Referencyjna metoda weryfikacji żywotności

W niniejszym rozdziale przedstawiono klasyczną metodę weryfikacji żywotności części sterującej systemu cyber-fizycznego (dalej nazywaną *metodą referencyjną*), bazującą na budowie oraz analizie grafu osiągalności. To właśnie do wyników uzyskanych za pomocą tej metody zostaną porównane wyniki osiągnięte z wykorzystaniem podejścia zaproponowanego w niniejszej rozprawie (rozdział 4).

Metoda referencyjna składa się z trzech głównych etapów, poglądowo przedstawionych na rys. 3.1.



Rysunek 3.1. Schemat metody referencyjnej

Pierwszym i zarazem kluczowym etapem metody referencyjnej jest wygenerowanie grafu osiągalności. Etap ten, wyróżniony na schemacie kolorem czerwonym, cechuje się wykładniczą złożonością obliczeniową, co w znacznym stopniu determinuje całkowitą efektywność algorytmu. W drugim etapie weryfikowana jest ograniczoność sieci, a w trzecim następuje weryfikacja żywotności w oparciu o wygenerowany graf osiągalności oraz komponenty silnie spójne. Należy w tym miejscu podkreślić, że

w literaturze weryfikacja *żywotności* jest często łączona z weryfikacją *odwracalności* sieci Petriego. Ponieważ niniejsza rozprawa skupia się typowo na aspektach żywotności, do dalszych rozważań przyjęto metodę weryfikacji żywotności sieci Petriego, bez uwzględnienia aspektów odwracalności. Ponadto, istotnym elementem omawianej metody jest *ograniczoność* sieci. W przypadku sieci nieograniczonych, dla których został wygenerowany graf pokrycia nie jest możliwe dokonanie weryfikacji żywotności. Dlatego też wynikiem działania metody w takim przypadku jest informacja, że sieć jest nieograniczona, a dalsza praca algorytmu jest przerywana.

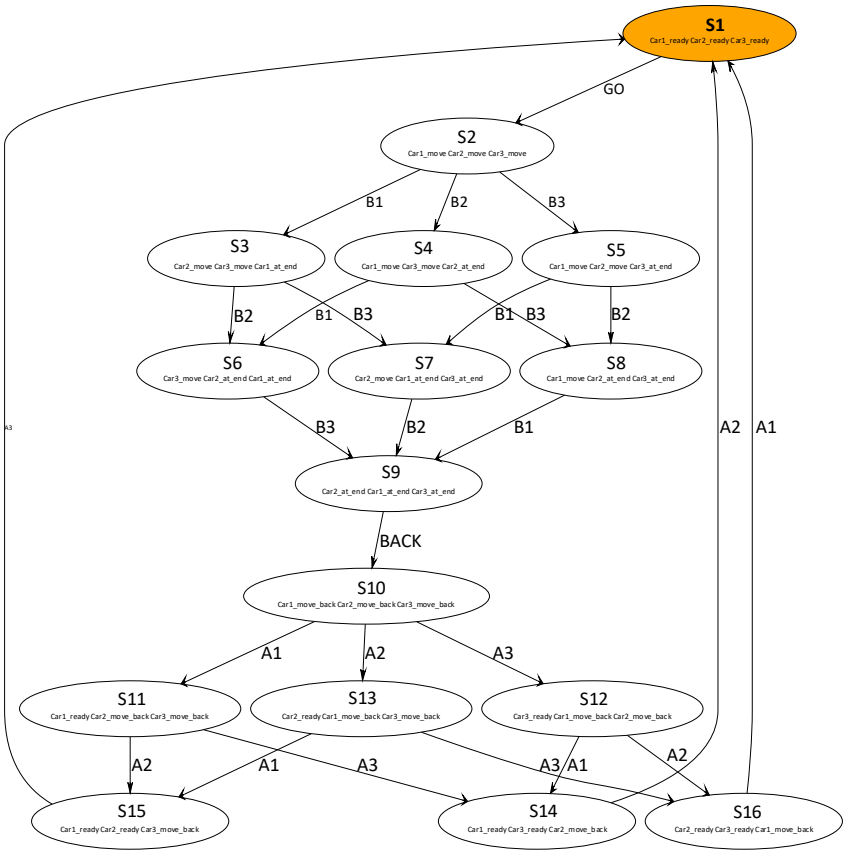
Kolejne podrozdziały przedstawiają szczegółowo etapy weryfikacji żywotności sieci. Najpierw omówiono mechanizm generowania grafu osiągalności, a następnie przedstawiono sposób jego analizy w celu określenia żywotności systemu opisanego siecią Petriego.

3.1. Generowanie grafu osiągalności

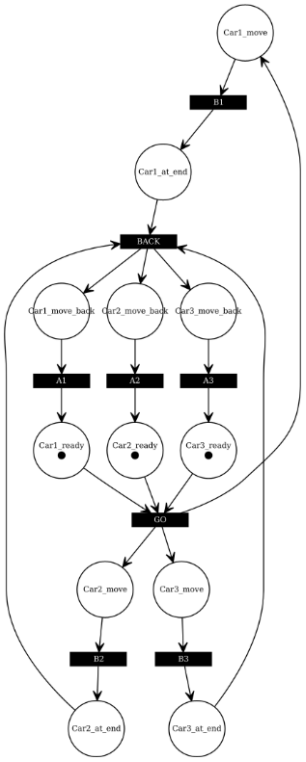
Formalną definicję grafu osiągalności przedstawiono w definicji 31. Niemniej, aby dokładnie wyjaśnić czym jest graf osiągalności i w jaki sposób jest wykorzystywany w procesie weryfikacji żywotności, poniżej zaprezentowano mniej formalny opis wraz przykładami. Warto w tym miejscu zaznaczyć, że w przypadku sieci nieograniczonej nie ma możliwości wygenerowania grafu osiągalności w skończonym czasie (generowanie grafu osiągalności dla sieci nieograniczonej trwałoby w nieskończoność). Dla sieci nieograniczonych możliwe jest wygenerowanie *grafu pokrycia* (algorytm do generowania grafu pokrycia jest wykorzystywany w algorytmie generowania grafu osiągalności, więc w przypadku sieci ograniczonej algorytm do generowania grafu pokrycia zwróci graf osiągalności). Niestety wykorzystanie grafu pokrycia do weryfikacji żywotności może dawać błędne wyniki, ponieważ technika, która umożliwia wygenerowanie skończonej liczby stanów powoduje utratę informacji o potencjalnych stanach martwych [27], [34], [67], [74].

Ogólnie rzecz ujmując, graf osiągalności składa się ze wszystkich możliwych stanów, w jakich może znaleźć się system opisany siecią Petriego. Budowa grafu rozpoczyna się od znakowania początkowego. Następnie analizowane są wszystkie możliwe znakowania, w jakich może znaleźć się system. Zgodnie z definicją 31, w grafie osiągalności węzłami są kolejne stany sieci Petriego, a krawędziami łuki etykietowane nazwami tranzycji, których odpalenie spowodowało przejście pomiędzy stanami. W grafie mogą znajdować

się różne ścieżki prowadzące do tego samego stanu. Na rys. 3.2 zaprezentowano graf osiągalności dla sieci Petriego o nazwie *3carros*, którą pokazano na rys. 3.3.



Rysunek 3.2. Graf osiągalności sieci *3carros*



Rysunek 3.3. Sieć Petriego o nazwie *3 carros*

Jak można zauważyć graf osiągalności składa się z 16 stanów oraz 26 łuków (rys. 3.2). W stanie S1 aktywnymi miejscami są *Car1_ready*, *Car2_ready*, *Car3_ready* (jest to jednocześnie stan początkowy sieci). Ze stanu S1 system może przejść do stanu S2 odpalając tranzycję *GO*. W Stanie S2 aktywnymi miejscami są *Car1_move*, *Car2_move* oraz *Car3_move*. W stanie S2 mogą zostać odpalone tranzycje *B1*, *B2* lub *B3*. Należy w tym miejscu zaznaczyć, że z założenia graf osiągalności opisuje zachowanie systemu, w których tranzycje są odpalane kolejno (nie jednocześnie), dlatego też analizowane są wszystkie możliwości. W przypadku odpalenia tranzycji *B1* generowany jest stan S3, odpalenie tranzycji *B2* prowadzi do stanu S4, a odpalenie tranzycji *B3* prowadzi do stanu S5. W stanie S3 aktywne są miejsca *Car2_move*, *Car3_move* oraz *Car1_at_end*. W przypadku stanu S4 aktywnymi miejscami są *Car1_move*, *Car3_move* oraz *Car2_at_end*. W stanie S5 miejscami, które posiadają token są *Car1_move*, *Car2_move* oraz *Car3_at_end*. Analizując graf osiągalności stanów można zauważyć, że w stanie S3 mogą zostać odpalone dwie tranzycje *B2* oraz *B3*. W stanie S4 aktywnymi tranzycjami są *B1* oraz *B3*, a ze stanu S5 mogą zostać odpalone tranzycje *B2* oraz *B1*.

Kontynuując analizę grafu osiągalności można zaobserwować, że stan S6 powstaje po odpaleniu dwóch różnych tranzycji. W przypadku stanów S7 oraz S8 jest podobnie. Do stanu S9 prowadzą trzy różne ścieżki. W tym fragmencie grafu można zauważyć, że występuje zrównoleglenie procesów (dotyczy to fragmentu od stanu S1 do stanu S9). W stanie S9 następuje ich synchronizacja, aby w kolejnym stanie S10 ponownie nastąpiło rozdzielenie procesów, których synchronizacja następuję w stanie S1 co zamyka analizę grafu osiągalności stanów.

Podczas analizy przedstawionego grafu osiągalności można zauważyć, że każde miejsce w trakcie postępu działania sieci stanie się aktywne. Nie wystąpi też sytuacja, w której w jakimkolwiek miejscu znajdzie się więcej niż jeden token (sieć jest ograniczona, a nawet bezpieczna). Ponadto, sieć wróci także do swojego stanu początkowego. Bardzo ważnym wnioskiem z analizy grafu osiągalności stanów jest również to, że z każdego miejsca w sieci można wyznaczyć skierowaną drogę powrotną do tego miejsca, na której wystąpią wszystkie inne miejsca z tej sieci (silna spójność).

Wszystkie opisane cechy, które można zaobserwować w grafie osiągalności świadczą o konkretnych właściwościach badanej sieci, takich jak bezpieczeństwo, odwracalność oraz żywotność sieci. W przypadku opisywanej sieci i jej grafu można dokonać analizy manualnie, ze względu na dość prostą strukturę grafu oraz stosunkowo mała liczbę stanów. W przypadkach większych sieci (dla których graf osiągalności stanów zawiera

dużą liczbę węzłów) taka manualna analiza jest bardzo trudna, a w wielu przypadkach wręcz niemożliwa, co wymusza wykorzystanie innych algorytmów, które na podstawie grafu osiągalności będą w stanie zweryfikować czy w badanej sieci występuje pożądana właściwość czy też nie.

Algorytm 1 przedstawia opisywaną metodę generowania grafu pokrycia (w przypadku sieci ograniczonej wynikiem tego algorytmu będzie graf osiągalności [27], [67]).

Algorytm 1. Generowanie grafu pokrycia dla sieci Petriego $\mathbf{N = (P, T, F, M_0)}$

1. Wczytaj sieć Petriego $\mathbf{N = (P, T, F, M_0)}$.
2. Przypisz do stanu $\mathbf{S_1}$ znakowanie początkowe sieci $\mathbf{M_0}$.
3. Dla danego węzła, który nie reprezentuje stanu martwego (stan martwy to taki, z którego nie można odpalić żadnej tranzycji), wygeneruj wszystkie stany $\mathbf{S_i}$ osiągalne przez odpalenie aktywnych tranzycji.
4. Jeżeli:
   - wygenerowany stan ma już swoją reprezentację w grafie – wówczas połącz ze sobą te stany;
   - w innym przypadku:
     - jeżeli w grafie występuje droga skierowana od stanu $\mathbf{S_1}$ do nowo wygenerowanego stanu $\mathbf{S_i}$ oraz istnieje węzeł reprezentujący stan $\mathbf{S_k}$, pokrywany przez $\mathbf{S_i}$, to utwórz nowy węzeł jako $\mathbf{S_j}$ taki, że $\mathbf{S_j(p) = S_i(p)}$, gdy $\mathbf{S_i(p) = S_k(p)}$ oraz $\mathbf{S_j(p) = \infty}$, gdy $\mathbf{S_i(p) > S_k(p)}$;
     - w innym przypadku: dołącz do grafu węzeł reprezentujący stan $\mathbf{S_i}$.
5. Do nowoutworzonych węzłów dołącz krawędzie etykietowane nazwą przejścia, którego odpalenie wygenerowało nowy stan.

3.2. Weryfikacja ograniczoności

W tym etapie weryfikowane jest, czy analizowana sieć Petriego jest ograniczona. Realizowane jest to na podstawie grafu osiągalności (pokrycia). Jeżeli po zakończeniu Algorytmu 1 w którymś ze stanów występuje symbol ∞, oznacza to, że sieć jest nieograniczona, a wygenerowany graf jest grafem pokrycia (a nie grafem osiągalności) i nie można go wykorzystać w następnym etapie badania żywotności. W takim przypadku algorytm kończy swoje działanie z informacją, że sieć jest nieograniczona.

W przypadku, gdy w żadnym ze stanów grafu osiągalności nie pojawia się symbol ∞, oznacza to, że sieć jest ograniczona i może być analizowana w kolejnym kroku metody referencyjnej. Opisywany Algorytm 1 pozwala wyznaczyć graf osiągalności tylko dla ograniczonych sieci Petriego. Głównym ograniczeniem Algorytmu 1 jest wykładnicza złożoność obliczeniowa, której przyczyną jest tzw. *eksplozja stanów* [27].

3.3. Weryfikacja żywotności

Do weryfikacji żywotności wykorzystywany jest graf osiągalności wygenerowany w poprzednim etapie. W tym celu analizowana jest *silna spójność* grafu. Warto w tym miejscu wspomnieć czym jest *silna spójność* oraz *komponenty silnie spójne*. Ich opis formalny znajduje się w definicji 12, niemniej warto doprecyzować jak można je wykorzystać do badania żywotności sieci Petriego. Komponenty silnie spójne (nazywane również *składowymi silnie spójnymi*) są podgrafami, których najważniejszą cechą jest to, że z każdego węzła (wierzchołka) danego komponentu można wyznaczyć drogę skierowaną (ścieżkę) do każdego innego węzła w tym komponencie [73]. Krótko mówiąc, istnieje możliwość przejścia pomiędzy dowolnymi dwoma stanami sieci. Przykładowo, dla prezentowanej w poprzednim rozdziale sieci *3carros* graf osiągalności zawiera jeden komponent silnie spójny. Komponent ten zawiera wszystkie stany co oznacza, że cały graf jest silnie spójny. To z kolei oznacza, że analizowana sieć jest żywa.

W prezentowanej metodzie do wygenerowania komponentów silnie spójnych wykorzystano algorytm naiwny (wynikający wprost z definicji), którego działanie opisuje *Algorytm 2* (poszczególne kroki bezpośrednio odzwierciadlają implementację algorytmu). W kontekście metody referencyjnej, wybór algorytmu naiwnego do identyfikacji silnie spójnych komponentów w grafie był świadomą i przemyślaną decyzją, podyktowaną celami jakie chciano osiągnąć w tej części pracy. Głównym założeniem metody referencyjnej było ustanowienie jasnego i zrozumiałego punktu odniesienia, umożliwiającego obiektywną ocenę efektywności proponowanego rozwiązania. Algorytm naiwny, choć nieoptymalny pod względem złożoności obliczeniowej, charakteryzuje się prostotą implementacji i intuicyjną logiką działania, co czyni go idealnym narzędziem do tego celu. Jego transparentność pozwala na łatwą weryfikację wyników i minimalizuje ryzyko wprowadzenia niejasności, które mogłyby utrudnić porównanie z bardziej zaawansowanym algorytmem zastosowanym w autorskim rozwiązaniu. Warto podkreślić, że autorowi rozprawy znane są alternatywne metody, takie jak algorytm Tarjana czy Kosaraju, które są powszechnie uznawane za bardziej efektywne w rozwiązywaniu tego

problemu (algorytm Tarjana znalazł nawet zastosowanie w autorskim rozwiązaniu, gdzie optymalizacja wydajności była kluczowym kryterium). Warto dodać, że wybór algorytmu naiwnego był wynikiem szerokiej dyskusji w zespole badawczym, w której uwzględniono zarówno aspekty teoretyczne, jak i praktyczne zastosowanego podejścia. Zdecydowano, że algorytm naiwny stanowi solidną i zrozumiałą bazę wyjściową, która pozwala na rzetelną ocenę korzyści wynikających z zastosowania bardziej zaawansowanego algorytmu w rozwiązaniu proponowanym w rozprawie. Ponadto, zastosowanie algorytmu naiwnego w metodzie referencyjnej pozwala na uwypuklenie zalet proponowanego rozwiązania, które demonstruje znaczną poprawę wydajności w porównaniu z prostym podejściem.

Algorytm 2. Generowanie komponentów silnie spójnych dla grafu osiągalności **G=(V, E)**

1. Wczytaj graf osiągalności sieci Petriego **G=(V, E)**.
2. Do listy **K** przypisz wszystkie węzły **V** grafu osiągalności **G**.
3. Do zmiennej **Indeks** przypisz wartość 0 (indeks pierwszego węzła z listy **K**).
4. Dla każdego elementu z listy **K** zaczynając od elementu o indeksie **Indeks + 1** (kolejne elementy z listy **K** oznaczono jako **k**) wykonuj następujące kroki:
   - **4a.** Sprawdź, czy istnieje skierowana droga pomiędzy elementem **k** a elementem z listy **K[Indeks]** (poszukiwanie takiej drogi można zrealizować za pomocą algorytmu przeszukiwania w głąb, ang. *Depth-First Search* – DFS);
   - **4b.** Sprawdź, czy istnieje skierowana droga pomiędzy elementem **K[Indeks]** a elementem **k**;
   - **4c.** Jeżeli nie istnieją skierowane drogi z kroku **4a** oraz **4b** należy usunąć element **k** z listy **K**.
5. Dla każdego elementu z listy **K** zaczynając od elementu o indeksie **Indeks + 2** wykonuj następujące kroki (kolejne elementy z listy **K** oznaczono jako **k'**):
   - **5a.** Sprawdź, czy istnieje skierowana droga pomiędzy elementem **k'** a elementem z listy **K[Indeks+1]**.
   - **5b.** Sprawdź, czy istnieje skierowana droga pomiędzy elementem **K[Indeks+1]** a elementem **k'**.
   - **5c.** Jeżeli nie istnieją skierowane drogi z kroku **5a** oraz **5b** należy usunąć element **k'** z listy **K**.

5d. Jeżeli wartość zmiennej **Indeks+2** jest mniejszy niż indeks ostatniego elementu listy **K**, zwiększ wartość zmiennej **Indeks** o jeden i przejdź do kroku **5**, w przeciwnym wypadku przejdź do kroku **6**.

6. Do listy **SCC** (lista **SCC** składa się z innych list) dopisz listę **K**.

7. Wyzeruj listę **K**.

8. Do listy **K** przypisz węzły z grafu osiągalności zaczynając od pierwszego węzła, który nie znajduje się w wyznaczonych wcześniej komponentach silnie spójnych.

9. Jeżeli lista **K** nie zawiera żadnego elementu **zakończ działanie algorytmu** (wszystkie komponenty silnie spójne znajdują się już na liście **SCC**) w przeciwnym przypadku przejdź do **kroku 3.**

Po wyznaczeniu komponentów silnie spójnych następuje etap weryfikacji żywotności. W tym celu należy w wyznaczonych komponentach silnie spójnych wyszukać komponent zawierający wszystkie tranzycje z badanej sieci Petriego oraz nie posiadający żadnego łuku wyjściowego. Brak takiego komponentu oznacza, że badana sieć nie jest żywa.

Propozycja 1: Złożoność obliczeniowa *Algorytmu 2* wynosi $O(|V|^3)$, gdzie $|V|$ jest liczbą wierzchołków (węzłów) w grafie osiągalności, dla których wyznaczane są komponenty silnie spójne.

Dowód: Zaprezentowany algorytm wyznacza kolejne silne spójne komponenty. W tym celu wykonywane są trzy bloki. Pierwszy z nich, realizowany przez pętlę główną (zewnętrzną, kroki 3-9), jest wykonywany $|V|$ razy. Oznacza to, że złożoność obliczeniowa tego bloku wynosi $O(|V|)$. Dwa kolejne bloki to pętle wykonywane wewnątrz pętli głównej. Pierwsza z nich (krok 4), jest wykonywana maksymalnie $|V|-1$ razy, wobec czego złożoność obliczeniowa jest szacowana jako $O(|V|)$. Druga z pętli wewnętrznych (krok 5) jest wykonywana $|V|-2$ razy i zawiera kolejną zagnieżdżoną pętlę wykonywaną $|V|-3$ razy. Oznacza to, że złożoność tego bloku (krok 5) jest szacowana jako $O(|V|^2)$. Reasumując, złożoność obliczeniowa całego *Algorytmu 2* wynosi $O(|V|^3)$, gdzie $|V|$ oznacza liczbę wierzchołków (węzłów) w grafie osiągalności, dla których wyznaczane są komponenty silnie spójne. ■

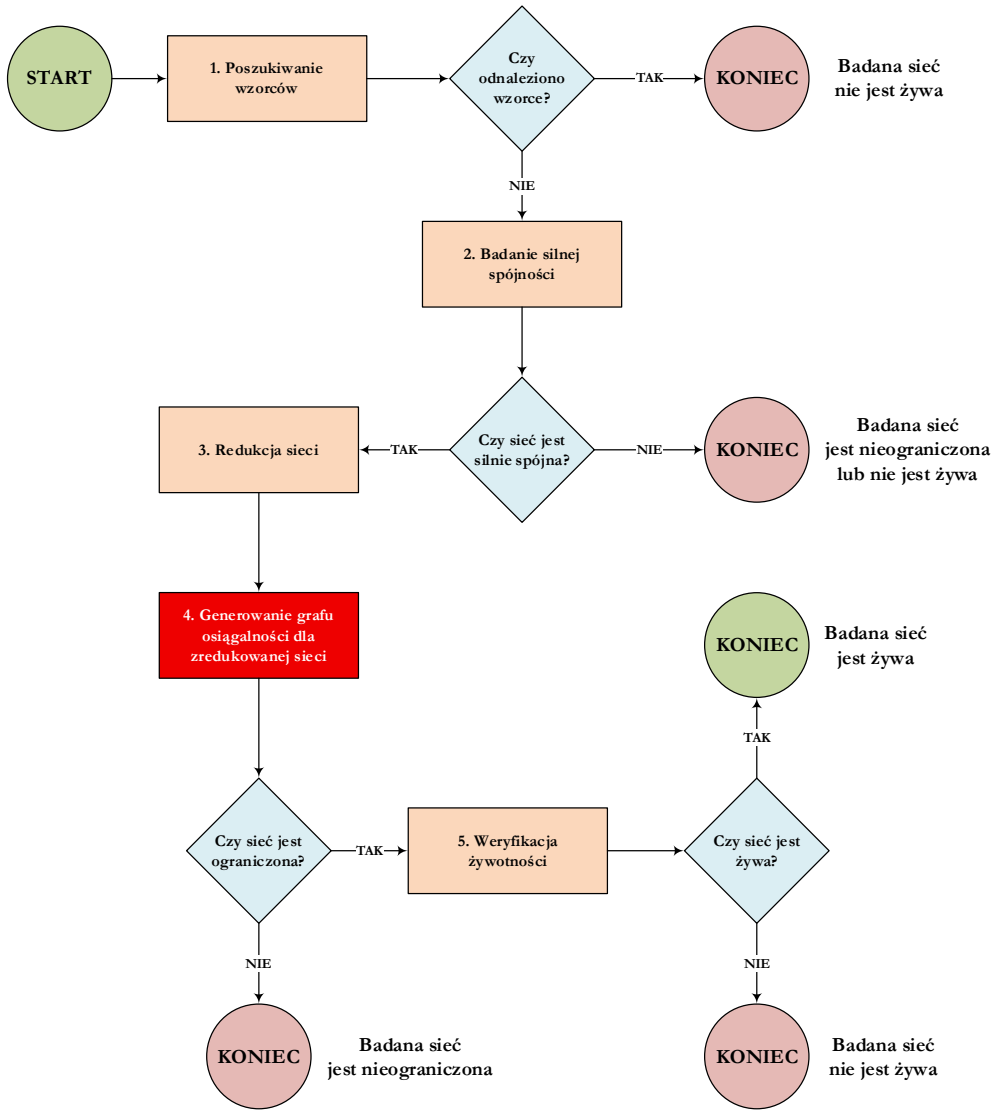
3.4. Podsumowanie

W rozdziale przedstawiono metodę referencyjną do weryfikacji żywotności sieci Petriego opisującej część sterującą systemu cyber-fizycznego. Zaprezentowano na schemacie kolejne etapy metody referencyjnej, a także opisano za pomocą kroków, w jaki sposób uzyskać wyniki w poszczególnych etapach metody. Przedstawiono na wybranym przykładzie zalety stosowania tego podejścia do weryfikacji żywotności sieci Petriego. Podjęto także próbę oszacowania złożoności obliczeniowej poszczególnych etapów metody referencyjnej. Pomimo, iż złożoność drugiego etapu jest wielomianowa, to głównym ograniczeniem całej metody jest etap pierwszy, którego złożoność jest wykładnicza. Dlatego też w niniejszej rozprawie zaprezentowano koncepcję alternatywną (rozdział 4), której celem jest efektywna weryfikacja żywotności części sterującej systemu cyber-fizycznego.

4. Proponowana metoda weryfikacji żywotności

W niniejszym rozdziale zaprezentowano autorską metodę weryfikacji żywotności części sterującej systemu cyber-fizycznego opisanej za pomocą sieci Petriego. Zgodnie z założeniami, opracowana koncepcja składa się z szeregu technik (pojedynczych metod), które tworzą kompletną metodę analizy żywotności. W pierwotnym zamyśle autora poszczególne techniki miały zostać przedstawione niezależnie, jednakże w wyniku prowadzonych prac okazało się, że znakomicie się uzupełniają. Dlatego też, ostatecznie zdecydowano się na opis kompletnej metody, natomiast poszczególne techniki zostaną omówione w ramach poszczególnych etapów tej metody.

Na rysunku 4.1 przedstawiono ogólny schemat proponowanego rozwiązania. W celu zwiększenia czytelności oraz przejrzystości, metodę podzielono na etapy (które, w dużym uproszczeniu, odpowiadają opracowanym technikom).



Rysunek 4.1. Schemat proponowanego rozwiązania

Ogólny sposób działania proponowanego podejścia jest następujący. Pierwszym etapem metody jest weryfikacja, czy w badanej sieci występują charakterystyczne wzorce (ang. *patterns*) dotyczące miejsc, tranzycji oraz połączeń między nimi. Tego typu wzorce pozwalają jednoznacznie stwierdzić, że analizowana sieć nie jest żywa. Jest to pierwsza z opracowanych, autorskich technik badania żywotności. Znalezienie jednego z takich wzorców od razu kończy proces weryfikacji. Natomiast, jeżeli w badanej sieci nie wystąpił żaden ze wzorców, następuje przejście do kolejnego etapu.

Drugim krokiem analizy jest weryfikacja silnej spójności sieci. Zgodnie z twierdzeniem [100] sieć, która nie jest silnie spójna nie może być żywa i ograniczona. Może to oznaczać, że sieć, która nie jest silnie spójna: (a) nie jest żywa, lub (b) jest nieograniczona, lub (c) nie jest żywa i jest nieograniczona. Uwzględniając fakt, że metoda weryfikacji żywotności z zastosowaniem grafu osiągalności wymaga sieci ograniczonych, jeżeli sieć nie jest silnie spójna, algorytm jest w tym miejscu przerywany. Należy jednocześnie podkreślić, że autor rozważał różne rozwiązania w sytuacji, gdy sieć nie jest silnie spójna. Alternatywnie, możliwa jest dalsza analiza sieci, aczkolwiek ma to sens jedynie w przypadku (a) tzn. gdy sieć jest ograniczona. Jednakże własność ta jest określana dopiero na etapie wyznaczania grafu osiągalności (jedyny krok o złożoności wykładniczej w proponowanej metodzie). Dlatego też, głównie ze względu na wyznaczony cel pracy doktorskiej (efektywność metody), ostatecznie zdecydowano się na bezwzględny wymóg silnej spójności sieci. W przypadku, gdy sieć nie jest silnie spójna, algorytm jest przerywany z informacją, że sieć nie jest żywa i/lub jest nieograniczona, a ewentualną dalszą analizę systemu pozostawiono w zakresie możliwych dalszych kierunków rozwoju metody. Badanie silnej spójności jest kolejną techniką zastosowaną przez autora rozprawy, a jej celem jest wstępna analiza części sterującej systemu cyber-fizycznego opisanej siecią Petriego.

W kolejnym, trzecim etapie realizowana jest iteracyjna redukcja sieci, z zastosowaniem zarówno autorskich, jak i istniejących technik redukcyjnych. Redukcja sieci zachowuje podstawowe własności sieci (w tym żywotność oraz ograniczoność), a jednocześnie może istotnie wpłynąć na skrócenie czasu analizy systemu.

W etapie czwartym generowany jest graf osiągalności dla zredukowanej sieci Petriego. W trakcie generowania tego grafu weryfikowana jest także ograniczoność zredukowanej sieci, co jest ważnym usprawnieniem w porównaniu do metody referencyjnej. W etapie końcowy proponowanego rozwiązania weryfikowana jest

żywotność z zastosowaniem autorskiej koncepcji analizy silnej spójności grafu osiągalności.

Trzy ostanie kroki prezentowanej metody (redukcja sieci, konstrukcja grafu osiągalności dla zredukowanej sieci oraz jego dalsza analiza) tworzą kolejną technikę badania żywotności sieci opisującej część sterującą systemu cyber-fizycznego, zastosowaną przez autora rozprawy. Jest to w pewnym sensie modyfikacja rozwiązania już istniejącego, w którym uwzględniono istotne usprawnienia (redukcję sieci, wczesne badanie ograniczoności oraz autorską metodę analizy skonstruowanego grafu).

Głównym celem opracowanej metody jest efektywna oraz skuteczna weryfikacja żywotności sieci Petriego. Na schemacie kolorem zielonym oznaczono etapy, których złożoność obliczeniowa jest wielomianowa, a kolorem czerwonym etap czwarty, dla którego złożoność obliczeniowa w najgorszym przypadku pozostaje wykładnicza. W tym miejscu należy podkreślić, że aż cztery etapy proponowanego rozwiązania mają wielomianową złożoność obliczeniową, z czego dwa stanowią techniki umożliwiające wstępną weryfikację sieci bez konieczności budowania grafu osiągalności. Jedyny etap, który ma złożoność wykładniczą dotyczy generowania grafu osiągalności, jednakże jest on konstruowany dla zredukowanej sieci, co ostatecznie może znacznie przełożyć się na redukcję czasu potrzebnego do weryfikacji sieci Petriego.

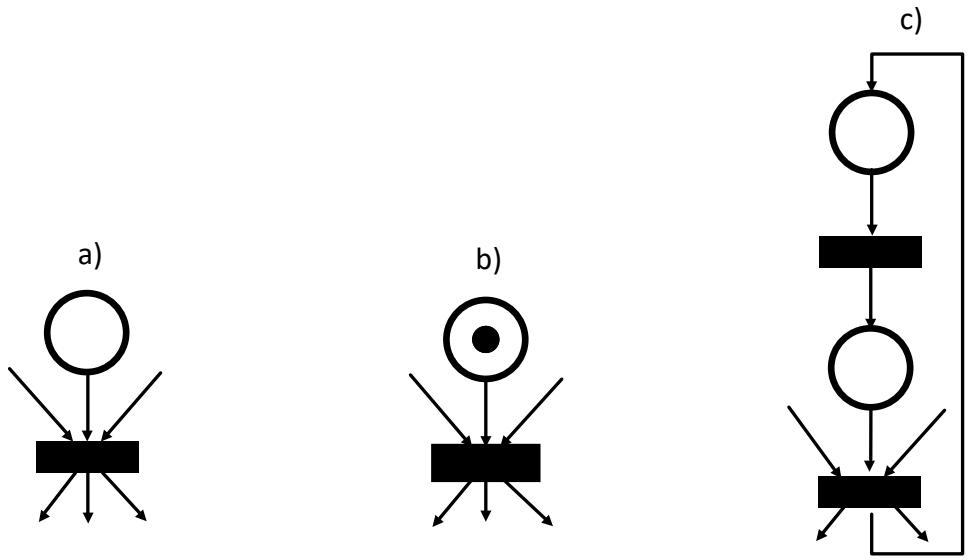
W kolejnych podrozdziałach szczegółowo omówiono poszczególne etapy proponowanej metody weryfikacji żywotności sieci Petriego (numeracja podrozdziałów odpowiada numerowi danego etapu metody).

4.1. Poszukiwanie wzorców

Poszukiwanie wzorców (ang. *pattern*) jest pierwszym etapem proponowanego rozwiązania. W literaturze przedmiotu spotkać można także określenie „sekwencja" zamiast „wzorzec", które mogą być interpretowane jako równoważne, dlatego w niniejszej pracy termin „sekwencja" używany jest zamiennie z terminem „wzorzec" – zwłaszcza w opisach algorytmów oraz ilustracjach. W opisywanym kroku proponowanej metody poszukiwane są trzy rodzaje sekwencji (rys. 4.2). Sekwencja pierwsza [101], [102] (w dalszej części pracy sekwencja będzie nazywana **S1**) składa się z jednego miejsca oraz jednej tranzycji (rys. 4.2a). Miejsce to nie posiada żadnego wejścia i jest połączone z tranzycją. Tranzycja, która wchodzi w skład sekwencji może posiadać wiele wejść oraz wyjść. Jedynym warunkiem jest to, aby jednym z wejść do tej tranzycji było ww. miejsce, które nie posiada żadnego wejścia.

Sekwencja druga **S2** jest wariantem sekwencji **S1**. Sekwencja **S2** [101], [102] co do struktury jest identyczna jak sekwencja **S1**, z tą różnicą, że w sekwencji **S2** miejsce bez wejścia zawiera co najmniej jeden token (rys. 4.2b).

Sekwencja trzecia **S3** składa się z dwóch miejsc oraz dwóch tranzycji. Miejsca w sekwencji **S3** nie mogą posiadać tokenów. Pierwsza tranzycja z sekwencji może posiadać tylko jedno wejście oraz jedno wyjście (łącząc dwa miejsca zawarte w sekwencji). Druga tranzycja może posiadać wiele wejść oraz wiele wyjść. Jedno z wejść do tej tranzycji musi być z drugiego miejsca w sekwencji a jedno z wyjść z tej tranzycji musi łączyć się z pierwszym miejscem w sekwencji (rys. 4.2c).



Rysunek 4.2. Poszukiwane sekwencje: a) S1, b) S2, c) S3

Do poszukiwania sekwencji opracowano autorskie algorytmy. W przypadku sekwencji **S1** oraz **S2** wykorzystywany jest ten sam algorytm z tą różnicą, że rozpatrywane miejsce wejściowe nie posiada (**S1**) lub posiada token (**S2**). Należy także podkreślić, że algorytm poszukiwania sekwencji jest przerywany w momencie wykrycia pierwszego wystąpienia sekwencji **S1** lub **S2** w sieci, co może skrócić czas poszukiwań. Na rysunku 4.3 przedstawiono algorytm poszukiwania sekwencji **S1** oraz **S2** w formie schematu blokowego. Wejściem algorytmu jest macierz incydencji sieci Petriego (oznaczona jako **A**), natomiast wyjściem jest informacja o znalezionych sekwencjach lub ich braku. *Algorytm 3* przedstawia poszczególne kroki opracowanej metody (w nawiasach podano skrócone nazwy zmiennych, które zostały wykorzystane w schemacie blokowym z rys. 4.3; kroki przedstawionego algorytmu są bezpośrednim odzwierciedleniem implementacji wykonanej przez autora rozprawy).

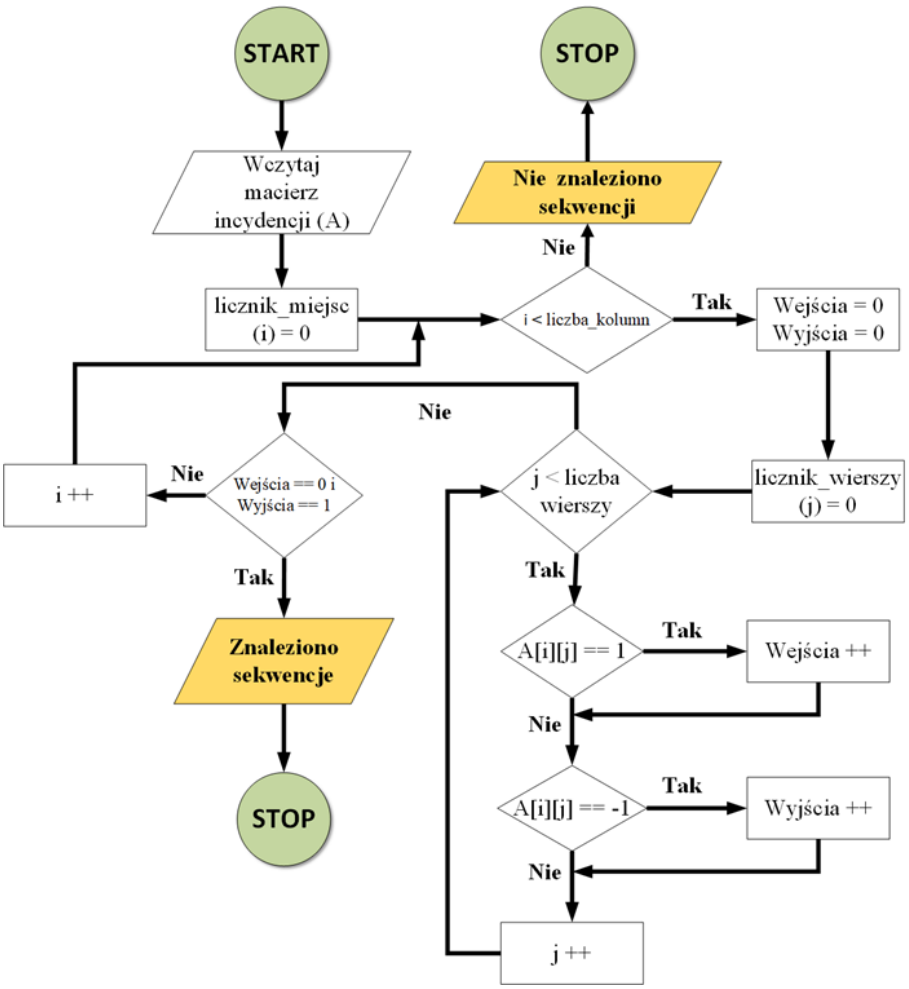
Algorytm 3. Wyszukiwanie sekwencji **S1** oraz **S2** w sieci Petriego $\mathbf{N = (P, T, F, M_0)}$

1. Wczytaj macierz incydencji **A[|P|][|T|]** sieci Petriego $\mathbf{N = (P, T, F, M_0)}$.
2. Przypisz do zmiennej **licznik miejsc (i)** wartość 0.
3. Jeżeli licznik miejsc jest mniejszy niż **|P|** to przejdź do **Kroku 4** w przeciwnym razie przejdź do **Kroku 12**.
4. Ustaw **licznik wejść** do miejsca **(wejścia)** oraz **licznik wyjść** z miejsca **(wyjścia)** na 0.
5. Ustaw **licznik wierszy (j)** na 0.
6. Jeżeli **licznik wierszy (j)** jest mniejszy od **|T|** to przejdź do **Kroku 7**, w przeciwnym razie przejdź do **Kroku 10**.
7. Jeżeli wartość elementu **A[i][j]** wynosi 1 to zwiększ **licznik wejść (wejścia)** o jeden.
8. Jeżeli wartość elementu **A[i][j]** wynosi -1 to zwiększ **licznik wyjść (wyjścia)** o jeden.
9. Zwiększ **licznik wierszy (j)** o jeden i przejdź do **Kroku 6**.
10. Jeżeli **licznik wyjść (wyjścia)** jest równy 1 oraz **licznik wejść (wejścia)** jest równy 0 to znaleziono sekwencję **S1** lub **S2** co **kończy algorytm**, w przeciwnym przypadku przejdź do **Kroku 11**.
11. Zwiększ **licznik miejsc (i)** o jeden i przejdź do **Kroku 3**.
12. Nie znaleziono sekwencji: **zakończ algorytm**.

Propozycja 2: Złożoność obliczeniowa *Algorytmu 3* wynosi **O(|P||T|)**, gdzie **|P|** oznacza liczbę miejsc, a **|T|** oznacza liczbę tranzycji sieci Petriego.

Dowód: Zaproponowany algorytm przeszukuje kolejne miejsca (zmienna **i** określająca licznik miejsc) oraz tranzycje (zmienna **j** określająca licznik tranzycji) pod kątem dopasowania określonych wzorców (**S1** oraz **S2**). W tym celu wykonywane są dwie pętle. Pierwsza z nich, to pętla zewnętrzna (kroki 3-11), która wykonuje się maksymalnie **|P|** razy, gdzie **|P|** oznacza liczbę wszystkich miejsc sieci. Wobec tego, złożoność obliczeniowa tej pętli wynosi **O(|P|)**. Z kolei druga, wewnętrzna pętla (kroki 6-9), jest wykonywana maksymalnie **|T|** razy, gdzie **|T|** oznacza liczbę tranzycji sieci Petriego. Złożoność obliczeniowa tej pętli wynosi **O(|T|)**. Reasumując, złożoność obliczeniowa

całego algorytmu oszacowano jako **O(|P||T|)** gdzie **|P|** oznacza liczbę miejsc, a **|T|** liczbę tranzycji sieci Petriego. ■



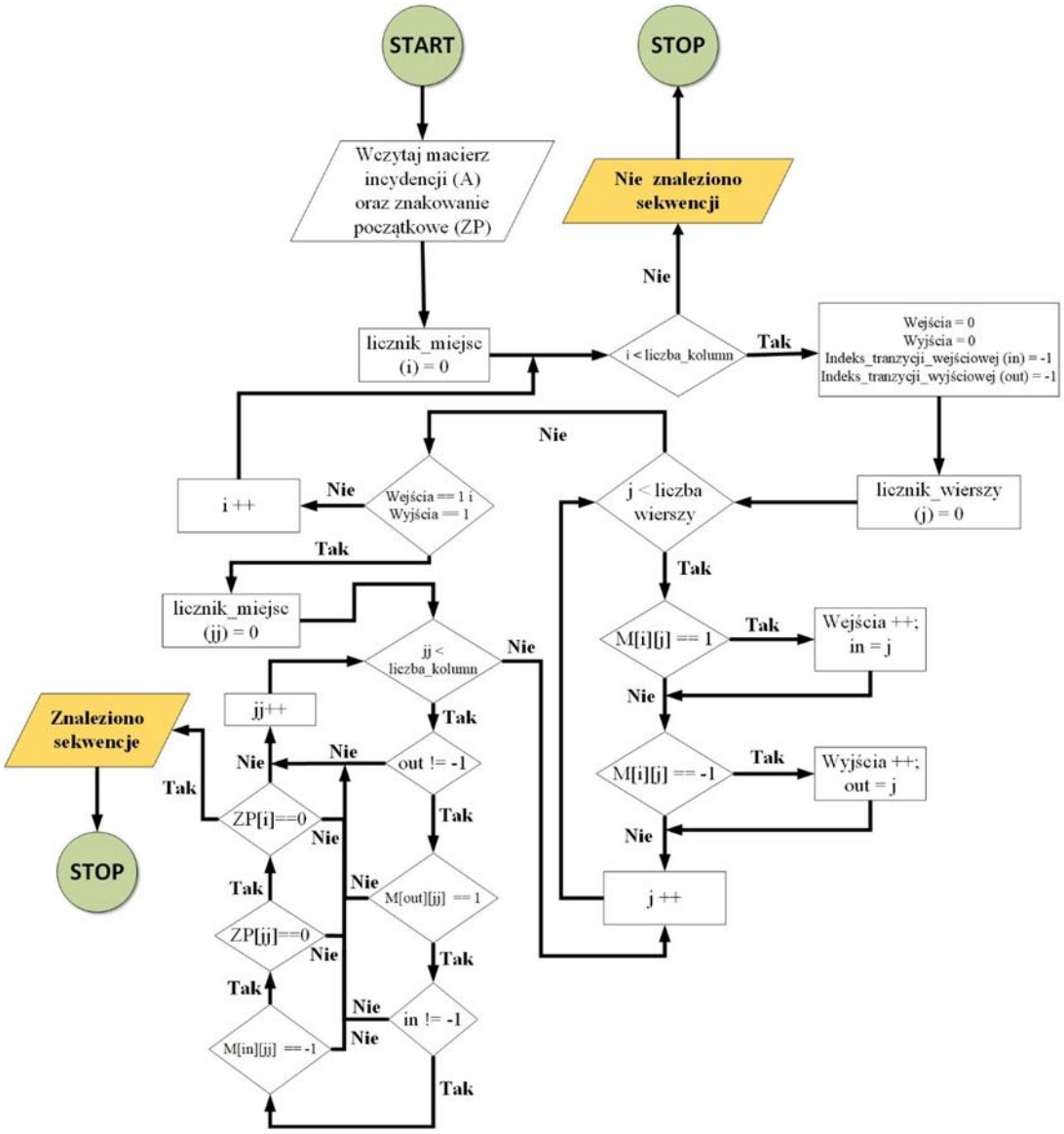
Rysunek 4.3. Schemat blokowy techniki poszukiwania sekwencji S1 oraz S2

Do poszukiwania sekwencji **S3** opracowano autorski *Algorytm 4* w którym przedstawiono poszczególne kroki proponowanej metody (w nawiasach podano skrócone nazwy zmiennych, które zostały wykorzystane w schemacie blokowym z rys. 4.4, przedstawione kroki są bezpośrednim odzwierciedleniem implementacji algorytmu zrealizowanego przez autora rozprawy, związku z tym dla zwiększenia czytelności znakowanie początkowe oznaczono skrótem **ZP** zamiast klasycznego M_0).

Algorytm 4. Wyszukiwanie sekwencji **S3** w sieci Petriego **N = (P, T, F, M_0)**

1. Wczytaj macierz incydencji **A[|P|] [|T|]** sieci Petriego **N = (P, T, F, ZP)**.
2. Przypisz do zmiennej **licznik miejsc (i)** wartość 0.
3. Jeżeli **licznik miejsc(i)** jest mniejszy niż liczba kolumn w macierzy incydencji to przejdź do **Kroku 4** w przeciwnym razie przejdź do **Kroku 15**.

4. Ustaw **licznik wejść** do miejsca **(wejścia)** oraz **licznik wyjść z miejsca (wyjścia)** na 0. Ustaw wartość **indeks tranzycji wejściowej (in)** na -1 oraz **indeks tranzycji wyjściowej (out)** na -1.
5. Ustaw **licznik wierszy (j)** na 0.
6. Jeżeli **licznik wierszy (j)** jest mniejszy od **|T|** to przejdź do **Kroku 7** w przeciwnym razie przejdź do **Kroku 10**.
7. Jeżeli wartość w macierzy incydencji **A[i][j]** wynosi 1 to zwiększ **licznik wejść (wejścia)** o jeden oraz ustaw **indeks tranzycji wejściowej (in) = licznik wierszy (j)**.
8. Jeżeli wartość w macierzy incydencji **A[i][j]** wynosi -1 to zwiększ **licznik wyjść (wyjścia)** o jeden oraz ustaw wartość **indeksu tranzycji wyjściowej (out)** na wartość **licznika wierszy (j)**.
9. Zwiększ **licznik wierszy (j)** o jeden i przejdź do **Kroku 6**.
10. Jeżeli **licznik wyjść (wyjścia)** jest równy 1 oraz **licznik wejść (wejścia)** jest równy 1 to przejdź do **Kroku 12** w przeciwnym wypadku przejdź do **Kroku 11**.
11. Zwiększ **licznik miejsc (i)** o jeden i przejdź do **Kroku 3**.
12. Ustaw **licznik miejsc(k)** na wartość 0.
13. Jeżeli **licznik miejsc(k)** jest mniejszy niż liczba kolumn w macierzy incydencji przejdź do **Kroku 14** w przeciwnym przypadku przejdź do **Kroku 9**.
14. Jeżeli **indeks tranzycji wyjściowej (out)** jest różny od wartości -1 oraz wartość w macierzy incydencji **A[out][k]** jest równa 1 oraz **indeks tranzycji wejściowej (in)** jest różny od -1 oraz wartość w macierzy incydencji **A[in][k]** jest równa -1 oraz miejsca o indeksach **i** oraz **k** nie są znakowane (**ZP[i] = 0** oraz **ZP[k] = 0**) to znaleziono sekwencję **S3**, co **kończy algorytm**, w przeciwnym przypadku zwiększ o jeden **licznik miejsc (k)** i przejdź do **Kroku 13**.
15. Nie znaleziono sekwencji: **zakończ algorytm**.



Rysunek 4.4. Schemat blokowy techniki poszukiwanie sekwencji S3

Propozycja 3: Złożoność obliczeniowa *Algorytmu 4* wynosi $O(|P|^2|T|)$, gdzie $|P|$ oznacza liczbę miejsc, a $|T|$ oznacza liczbę tranzycji sieci Petriego.

Dowód: Zaproponowany algorytm przeszukuje kolejne miejsca (zmienne i oraz k określają liczniki miejsc) oraz tranzycje (zmienna j określa licznik tranzycji) pod kątem dopasowania wzorca $S3$. W tym celu wykonywane są trzy pętle. Pierwsza z nich, to pętla zewnętrzna (kroki 3-14), która wykonuje się maksymalnie $|P|$ razy, gdzie $|P|$ oznacza liczbę wszystkich miejsc sieci. Wobec tego, złożoność obliczeniowa tej pętli wynosi $O(|P|)$. Z kolei druga, wewnętrzna pętla (kroki 6-9), jest wykonywana maksymalnie $|T|$ razy, gdzie $|T|$ oznacza liczbę tranzycji sieci Petriego. Złożoność obliczeniowa tej pętli wynosi $O(|T|)$. Dodatkowo, w tym bloku wykonywana jest kolejna zagnieżdżona pętla wewnętrzna operująca na zmiennej k. Pętla ta jest wykonywana maksymalnie $|P|$ razy,

wobec czego jej złożoność obliczeniowa wynosi $O(|P|)$. Reasumując, złożoność obliczeniowa całego algorytmu jest szacowana jako $O(|P|^2|T|)$ gdzie $|P|$ oznacza liczbę miejsc, a $|T|$ liczbę tranzycji sieci Petriego. ■

Odnalezienie sekwencji **S1**, **S2** oraz **S3** oznacza, że sieć nie jest żywa. W przypadku sekwencji **S1** tranzycja, która jest w niej zawarta nie będzie odpalona ani razu w trakcie jej działania. Wynika to z faktu, że w miejscu wejściowym do tej tranzycji nigdy nie znajdzie się token, który jest niezbędny, aby tranzycja została odpalona (zgodnie z definicją 24, aby tranzycja mogła być odpalona to w każdym miejscu wejściowym do tej tranzycji musi znajdować się przynajmniej jeden token).

Sekwencja **S2** co prawda umożliwi odpalenie tranzycji, ale tylko tyle razy, ile tokenów w znakowaniu początkowym znajduje się w miejscu wejściowym do tej tranzycji. Po wyczerpaniu się tokenów nastąpi sytuacja analogiczna jak w sekwencji **S1**. Takie zachowanie również uniemożliwia, aby sieć była żywa (z definicji, aby sieć była L4-żywa, wszystkie tranzycje muszą być żywe, czyli muszą być możliwe to odpalenia nieskończenie wiele razy – w przypadku sekwencji **S2** tranzycja w niej zawarta nie będzie żywa, co tym samym oznacza, że cała sieć też nie będzie żywa).

W przypadku sekwencji **S3** sytuacja jest bardzo podobna, bowiem w miejscach zawartych w tej sekwencji nigdy nie pojawią się tokeny, co tym samym uniemożliwia, aby zawarte w niej tranzycje mogły zostać odpalone (nawet jeżeli tranzycje mają więcej wejść oraz wyjść niż oznaczono na rysunku 4.2).

Warto zauważyć, że w wielu przypadkach algorytm może być przerwany wcześniej, ponieważ pierwsze wystąpienie sekwencji powoduje natychmiastowe zatrzymanie algorytmu (pojedyncze wystąpienie sekwencji **S1**, **S2** lub **S3** oznacza brak możliwości, aby sieć była żywa). Reasumując, poszukiwanie wszystkich rodzajów sekwencji odbywa się wg następującej procedury:

1. Weryfikacja wystąpienia sekwencji **S1** oraz **S2**, gdzie wystąpienie sekwencji **S1** lub **S2** skutkuje **zakończeniem działania metody** (sieć nie jest żywa), w przeciwnym przypadku przejdź do **Kroku 2**.
2. Weryfikacja wystąpienia sekwencji **S3**, gdzie w przypadku wystąpienia sekwencji **S3** skutkuje **zakończeniem działania metody** (sieć nie jest żywa), w przeciwnym przypadku **przejdź do następnego etapu** proponowanej metody (badanie silnej spójności).

4.2. Badanie silnej spójności

W kolejnym etapie badana jest silna spójność sieci Petriego. Proponowana metoda jest zmodyfikowaną wersją koncepcji pierwotnie pokazanej w [103]. Wersja oryginalna umożliwia poszukiwanie komponentów silnie spójnych, natomiast algorytm użyty w pracy jest ściśle zorientowany na weryfikację silnej spójności całej sieci (określany jest jeden komponent, który musi być silnie spójny i zawierać wszystkie wierzchołki sieci, tzn. miejsca i tranzycje). Metoda wykorzystuje przeszukiwanie w głąb (*ang. Depth-First Search* – DFS) w wersji rekurencyjnej, w trakcie którego numeruje odwiedzone wierzchołki. *Algorytm 5* przedstawia kroki, z których składa się wykorzystana na tym etapie metoda badania silnej spójności.

Algorytm 5. Badanie silnej spójności sieci Petriego $\mathbf{N = (P, T, F, M_0)}$

1. Wczytaj sieć Petriego $\mathbf{N = (P, T, F, M_0)}$.
2. Do zmiennej $\mathbf{V}$ przypisz sumę zbiorów $\mathbf{P}$ i $\mathbf{T}$ ($\mathbf{V = P \cup T}$).
3. Zainicjalizuj stos $\mathbf{S}$ zbiorem pustym ($\mathbf{S \leftarrow \emptyset}$).
4. Zmienną $\mathbf{SCC}$ zainicjalizuj zbiorem pustym ($\mathbf{SCC \leftarrow \emptyset}$) (w zmiennej $\mathbf{SCC}$ zostanie zapisany silnie spójny komponent).
5. Do zmiennej **indeks** przypisz wartość **0** (**indeks = 0**).
6. Zainicjalizuj wektor **lowlink[|V|]** wartościami **-1**.
7. Zainicjalizuj wektor **44rof.[|V|]** wartościami **-1**.
8. Dla dowolnego wierzchołka $\mathbf{v \in V}$ wykonaj funkcje **silniePolaczony(v).**
9. Jeżeli $\mathbf{SCC = V}$ oznacza, że sieć jest **silnie spójna** w przeciwnym przypadku sieć **nie jest silnie spójna. Zakończ algorytm.**

Funkcja **silniePolaczony(v ∈V)** składa się z następujących kroków:

1. Do wektora **lowlink** o indeksie **v** przypisz wartość zmiennej **indeks (lowlink[v] = indeks)**.
2. Do wektora **44rof.** o indeksie **v** przypisz wartość zmiennej **indeks (44rof.[v] = indeks**).
3. Zwiększ wartość zmiennej **indeks** o **jeden** (**indeks = indeks + 1**).
4. Na stosie $\mathbf{S}$ umieść $\mathbf{v}$ ($\mathbf{S = S \cup \{v\}}$).
5. Dla każdego $\mathbf{w}$ należącego do $\mathbf{V}$ ($\mathbf{w \in V}$), który jest sąsiadem $\mathbf{v}$ wykonaj:
   a. Jeżeli **44rof.[w]** równe jest **-1** (**44rof.[w] == -1**) to:
      - Wywołaj funkcję **silniePolaczony(w).**

- Do wektora **lowlink** o indeksie **v** przypisz wartość minimum z **lowlink[v]** oraz **lowlink[w]** (**lowlink[v] = min(lowlink[v], lowlink[w])**).

b. W przeciwnym razie:

- Jeżeli **45rof.[w]** jest mniejsze od **45rof.[v]** oraz **w** należy do stosu **S** (**45rof.[w]<45rof.[v]** oraz **w ∈ S**) to:
  - Do wektora **lowlink** o indeksie **v** przypisz wartość minimum z **lowlink[v]** oraz **lowlink[w]** (**lowlink[v] = min(lowlink[v], lowlink[w])**).

6. Jeżeli **lowlink[v]=45rof.[v]** to:

a. Do zmiennej **SCC** dodaj wszystkie wartości ze stosu **S** (**SCC = SCC ∪ S**) oraz przypisz pusty zbiór do stosu **S** (**S←∅**).

Wspomiana wyżej modyfikacja algorytmu polegała na wyznaczeniu silnie spójnego komponentu tylko dla jednego wierzchołka **v ∈ V**, a nie tak jak w wersji oryginalnej dla każdego nieodwiedzonego wierzchołka. Takie podejście umożliwia wyznaczenie jednego silnie spójnego komponentu i sprawdzeniu czy zawiera on wszystkie miejsca oraz tranzycje. Jeżeli ten komponent silnie spójny nie zawiera wszystkich miejsc oraz tranzycji oznacza to, że sieć nie jest silnie spójna i nie ma potrzeby wyznaczać kolejnych silnie spójnych komponentów sieci.

Propozycja 4: Złożoność obliczeniowa *Algorytmu 5* wynosi **O(|P|+|T|+|F|)**, gdzie **|P|** oznacza liczbę miejsc, **|T|** oznacza liczbę tranzycji, a **|F|** liczbę łuków sieci Petriego.

Dowód: na podstawie dowodu przedstawionego w publikacji [103] wiemy, że złożoność obliczeniowa algorytmu poszukującego wszystkie komponenty silnie spójne w grafie jest liniowa względem liczby wierzchołków oraz krawędzi grafu **O(|V|+|E|)**. Sieć Petriego jest grafem, w którym zbiór wszystkich wierzchołków to suma zbiorów miejsc **P** oraz tranzycji **T** (czyli **V=P ∪ T**), z kolei krawędzie opisane są poprzez łuki **F**. Wprowadzone usprawnienie dotyczy kroku 9, w którym sprawdzany jest końcowy wynik. Jest to pojedyncza operacja, która nie wpływa na całkowitą złożoność obliczeniową. Reasumując, złożoność obliczeniowa *Algorytmu 5* wynosi **O(|P|+|T|+|F|)**, gdzie **|P|** oznacza liczbę miejsc, **|T|** oznacza liczbę tranzycji, a **|F|** liczbę łuków sieci Petriego. ■

W przypadku, gdy badana sieć jest silnie spójna, metoda przechodzi do kolejnego kroku weryfikacji żywotności. W przeciwnym razie algorytm zwraca informację, że analizowana sieć nie jest żywa i/lub jest nieograniczona.

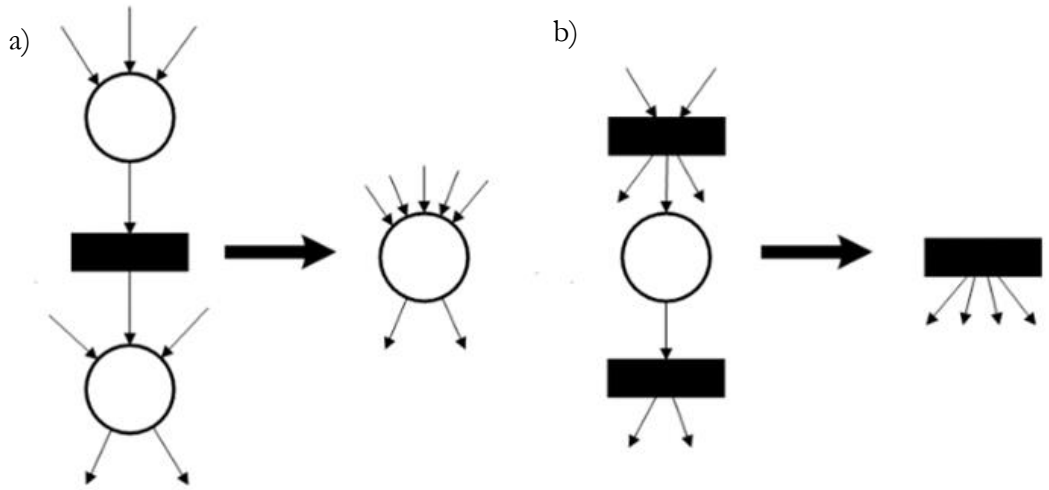
4.3. Redukcja sieci

Kolejnym etapem proponowanej metody jest wykorzystanie technik redukcyjnych, które z jednej strony znacząco potrafią zredukować sieć (co jest istotne z punktu widzenia efektywności analizy), a z drugiej zapewniają zachowanie właściwości oryginalnej sieci. W tym etapie wykorzystano zarówno techniki znane, jak i autorską metodę redukcji. Redukcjami zaczerpniętymi wprost z literatury są [27], [104]:

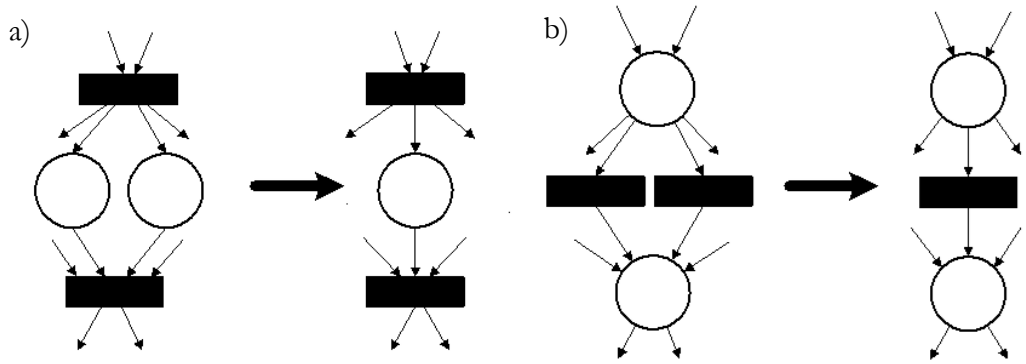
- scalanie kolejnych miejsc (ang. *Fusion of Series Places* – FSP),
- scalanie kolejnych tranzycji (ang. *Fusion of Series Transitions* – FST),
- scalanie równoległych miejsc (ang. *Fusion of Parallel Places* – FPP),
- scalanie równoległych tranzycji (ang. *Fusion of Parallel Transitions* – FPT).

Z kolei autorską, opracowaną redukcję nazwano *scalaniem miejsc i usuwaniem ciasnych pętli miejsc* (ang. *Fusion of places and Elimination of Self-loop Places* – FESP). Jest to swoiste połączenie redukcji FSP z jednoczesnym usuwaniem ciasnych pętli miejsc (ang. *Elimination of self-loop places* – ESP). Metoda ta została opracowana ze względu na format wejściowy wykorzystywany przez algorytm. Wejściem metody jest macierz incydencji sieci Petriego, w której nie można zapisać tzw. *ciasnych pętli*. Wprowadzone rozwiązanie znacznie rozszerzyło zastosowanie możliwości technik redukcji, gdyż w przypadku klasycznych metod *ciasna pętla* uniemożliwia przeprowadzenie operacji redukcji.

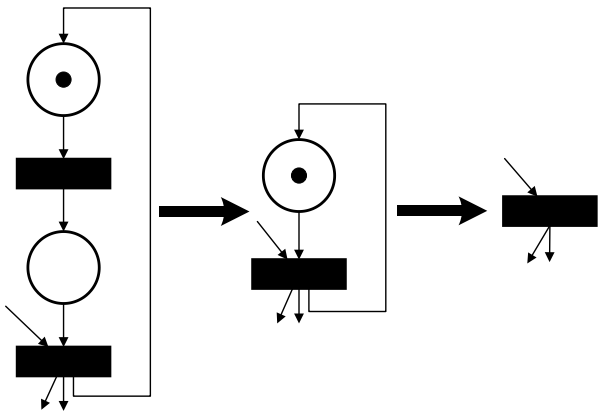
Proces redukcji w proponowanym rozwiązaniu wykonywany jest cyklicznie. Redukcje realizowane są w następującej kolejności: FSP, FST, FPP, FPT. Po każdym cyklu sprawdzane jest, czy zmniejszyła się liczba miejsc oraz tranzycji. Cykle redukcji są przerywane, jeśli po zakończeniu cyklu i sprawdzeniu czy liczba miejsc oraz tranzycji nie została zmniejszona. Po wykonaniu redukcji cyklicznej wykonywana jest redukcja FESP. Na rysunku 4.5 przedstawiono redukcje FSP oraz FST, na rysunku 4.6 redukcje FPP oraz FPT, a na rysunku 4.7 redukcję FESP.



Rysunek 4.5. Schemat redukcji a) FSP ora b) FST



Rysunek 4.6. Schemat redukcji: a) FPP oraz b) FPT



Rysunek 4.7. Schemat redukcji FESP

Główną korzyścią z zastosowania redukcji jest zmniejszenie liczby miejsc, tranzycji oraz połączeń między nimi w sieci Petriego, co przekłada się na późniejszą mniejszą liczbę stanów w grafie osiągalności stanów, którego generowanie następuje w następnym etapie proponowanego rozwiązania.

4.4. Generowanie grafu osiągalności dla zredukowanej sieci

Kolejnym etapem proponowanej metody jest generowanie grafu osiągalności dla zredukowanej sieci Petriego. Do tego wykorzystano zmodyfikowaną wersję *Algorytmu 1* opisanego w rozdziale związanych z metodą referencyjną. W *Algorytmie 6* zaprezentowano kroki, które zostały wykorzystane w tym etapie autorskiej metody.

Algorytm 6. Generowanie grafu osiągalności (zmodyfikowana wersja) dla sieci Petriego $\mathbf{N = (P, T, F, M_0)}$

1. Wczytaj sieć Petriego $\mathbf{N = (P, T, F, M_0)}$.
2. Przypisz do stanu $\mathbf{S_1}$ znakowanie początkowe sieci $\mathbf{M_0}$.
3. Dla danego węzła, który nie reprezentuje stanu martwego (stan martwy to taki, z którego nie można odpalić żadnej tranzycji), wygeneruj wszystkie stany $\mathbf{S_i}$ osiągalne przez odpalenie aktywnych tranzycji.
4. Jeżeli:
   a. wygenerowany stan ma już swoją reprezentację w grafie – wówczas połącz ze sobą te stany;
   b. w innym przypadku: jeżeli w grafie występuje droga skierowana od stanu $\mathbf{S_1}$ do nowo wygenerowanego stanu $\mathbf{S_i}$ oraz istnieje węzeł reprezentujący stan $\mathbf{S_k}$, pokrywany przez $\mathbf{S_i}$, to sprawdź czy nowo utworzony węzeł $\mathbf{S_j}$ taki, że $\mathbf{S_j(p) = S_i(p)}$, gdy $\mathbf{S_i(p) = S_k(p)}$ oraz $\mathbf{S_j(p) = \infty}$, gdy $\mathbf{S_i(p) > S_k(p)}$; będzie posiadał symbol $\mathbf{\infty}$, jeżeli tak będzie przerwij działanie algorytmu (sieć jest nieograniczona);
   c. w innym przypadku: dołącz do grafu węzeł reprezentujący stan $\mathbf{S_i}$.
5. Do nowoutworzonych węzłów dołącz krawędzie etykietowane nazwą tranzycji, której odpalenie wygenerowało nowy stan.

Różnica w tym algorytmie w stosunku do użytego w metodzie referencyjnej jest w kroku generowania nowego stanu. W przypadku algorytmu wykorzystanego w proponowanym rozwiązaniu już na etapie generowania kolejnych stanów w grafie weryfikowana jest ograniczoność, która jest niezbędna w dalszych etapach proponowanego rozwiązania. Wykrycie pierwszego miejsca, które jest nieograniczone przerywa dalsze generowanie grafu pokrycia, co również kończy działanie proponowanego rozwiązania z informacją, że badana sieć jest nieograniczona. Jest to

znaczące usprawnienie w porównaniu do metody referencyjnej, która wymaga określenia pełnego grafu i dopiero na jego podstawie analizowana jest ograniczoność sieci.

Jeżeli w trakcie generowania kolejnych stanów nie wystąpiło miejsce nieograniczone to wynikiem działania algorytmu będzie graf osiągalności, który zostanie wykorzystany w kolejnym etapie proponowanego rozwiązania. Tak jak w przypadku metody referencyjnej tak i tutaj głównym ograniczeniem tego etapu jest wykładnicza złożoność obliczeniowa w ogólnym przypadku, której przyczyną tak jak w metodzie referencyjnej jest tzw. „eksplozja przestrzeni stanów". Należy jednak tutaj podkreślić, że na tym etapie proponowanej metody graf osiągalności (graf pokrycia) jest generowany dla już zredukowanej sieci, co znacząco ogranicza wystąpienie tego problemu.

4.5. Weryfikacja żywotności

Po przygotowaniu grafu osiągalności następuje właściwa procedura weryfikacji czy badana sieć jest żywa. Wynik takiej weryfikacji jest dokładny i nie jest obarczony błędem. Na rysunku 2.7 przedstawiono graf osiągalności dla jednej z testowych sieci *3carros* z bazy *Hippo* wykorzystanych do weryfikacji opracowanych metod.

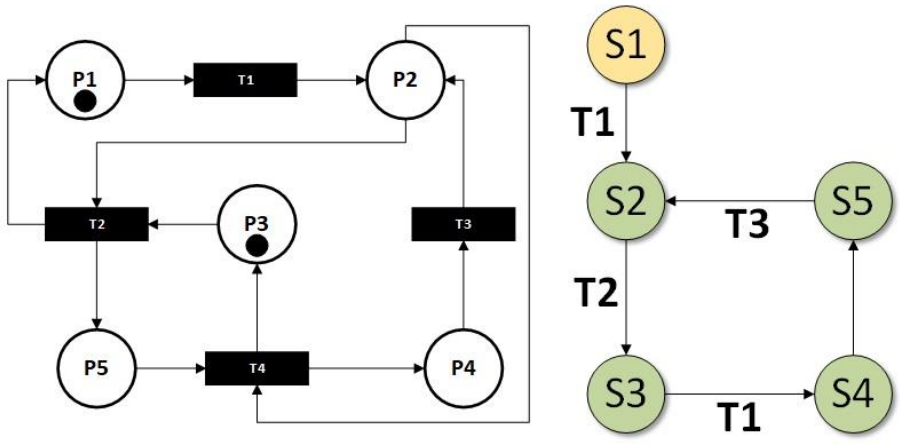
W tym etapie wykorzystywany jest algorytm do generowania silnie spójnych komponentów z grafu osiągalności (wykorzystano tu zmodyfikowany *Algorytm 5* do weryfikacji silnej spójności sieci Petriego z rozdziału 4.2). W *Algorytmie 5* wyznaczany jest tylko jeden silny spójny komponent, a w tym etapie wyznaczane są kolejne komponenty silnie spójne (wyznaczanie zostanie przerwane w momencie spełniania warunku, który zostanie opisany w dalszej części rozdziału).

Kolejna modyfikacja, która została wprowadzona w algorytmie jest to, co zawiera wyznaczony silnie spójny komponent. W oryginalnym *Algorytmie 5* w skład silnie spójnego komponentu wchodziły wierzchołki (w rozdziale 4.2 wierzchołkami były miejsca oraz tranzycje), a w tej wersji w skład silnie spójnych komponentów wchodzą łuki etykietowane nazwami tranzycji z grafu osiągalności. W procesie wyznaczania kolejnych silnie spójnych komponentów weryfikowane jest również, czy komponent zawiera wszystkie etykiety tranzycji z sieci. Dodatkowo, komponent ten musi być końcowy, tzn. nie może posiadać łuków wychodzących do innych wierzchołków spoza komponentu. Jeżeli nowo wygenerowany komponent zawiera wszystkie etykiety tranzycji z sieci i jest komponentem końcowym, oznacza to, że sieć jest żywa co kończy generowanie kolejnych silnie spójnych komponentów. *Algorytm 7* przedstawia proponowaną metodę weryfikacji żywotności systemu na podstawie grafu osiągalności sieci Petriego.

Algorytm 7. Weryfikacja żywotności na podstawie grafu osiągalności **G=(V, E)**

1. Wczytaj graf osiągalności sieci Petriego **G=(V, E)**.
2. Rozpocznij wyznaczanie silnie spójnych komponentów:
   a. Wyznacz silnie spójny komponent zawierający etykiety łuków grafu osiągalności (nazwy tranzycji);
   b. Zweryfikuj czy wyznaczony silnie spójny komponent zawiera wszystkie tranzycje z sieci oraz jest komponentem końcowym. Jeżeli spełnione są te oba warunki to sieć jest **żywa** co **kończy algorytm**, w przeciwnym razie przejdź do kroku **2a**.
3. Jeżeli żaden z wyznaczonych silnie spójnych komponentów nie spełnia warunku z kroku **2a** oznacza to, że badana sieć **nie jest żywa**, co **kończy algorytm**.

Powyższa modyfikacja algorytmu była konieczna, ponieważ sprawdzenie czy graf osiągalności jest tylko silnie spójny weryfikuje żywotność oraz odwracalność sieci [67], [77]. Istotnie, bardzo często obie wymienione właściwości występują wspólnie w sieci, jednakże również można odnaleźć w literaturze przykłady sieci, które nie są odwracalne, a nadal są żywe (co w przypadku weryfikacji samej silnej spójności grafu osiągalności nie gwarantuje potwierdzenia żywotności badanej sieci). Na rysunku 4.8 przedstawiono przykład takiej sieci (sieć nie jest odwracalna, ale żywa i ograniczona) oraz jej graf osiągalności.



Rysunek 4.8. Przykładowa sieć Petriego oraz jej graf osiągalności

Propozycja 5: Złożoność obliczeniowa *Algorytmu 7* wynosi **O(|V|+|E|)**, gdzie **|V|** oznacza liczbę stanów (znakowań osiągalnych) zredukowanej sieci, a **|E|** oznacza liczbę łuków w grafie osiągalności.

Dowód: Na podstawie dowodu przedstawionego w publikacji [103] wiemy, że złożoność obliczeniowa algorytmu poszukującego wszystkie komponenty silnie spójne w grafie jest liniowa względem liczby wierzchołków oraz krawędzi grafu $O(|V|+|E|)$. Graf osiągalności sieci Petriego jest grafem, w którym zbiór wszystkich wierzchołków według *Definicji 1* jest zbiorem wszystkich osiągalnych stanów ze znakowania początkowego M_0. Z kolei krawędzie opisane poprzez łuki E są zbiorem łuków etykietowanych nazwami tranzycji. Wprowadzone modyfikacje (w stosunku do *Algorytmu 3* z rozdziału 4.2), dotyczą sprawdzenia końcowego wyniku. Są to pojedyncze operacje, które nie mają wpływu na całkowitą złożoność obliczeniową algorytmu. Reasumując, złożoność obliczeniowa *Algorytmu 7* wynosi $O(|V|+|E|)$, gdzie $|V|$ oznacza liczbę stanów, a $|E|$ oznacza liczbę łuków grafu osiągalności zredukowanej sieci Petriego. ■

Należy w tym miejscu podkreślić, że liczba stanów zredukowanej sieci jest skończona, gdyż w przypadku sieci nieograniczonych algorytm przerywa działanie już na etapie konstrukcji grafu.

5. Eksperymentalna weryfikacja zaproponowanego rozwiązania

W niniejszym rozdziale omówione zostaną wyniki badań eksperymentalnych opracowanych metod weryfikacji żywotności. Do weryfikacji użyto zestawu 246 benchmarków (testów) z bazy systemu *Hippo*. Opracowaną metodę (wraz ze wszystkimi wchodzącymi w jej skład technikami), a także metodę referencyjną autor zaimplementował z zastosowaniem języka C++. Następnie, przeprowadzono badania eksperymentalne mające na celu potwierdzenie efektywności oraz skuteczności opracowanej metody badania żywotności (w porównaniu do metody referencyjnej). Należy w tym miejscu podkreślić, że w badaniach weryfikowana była pełna metoda, a nie jej poszczególne składowe (poszczególne techniki), gdyż autor rozprawy uznał, że takie podejście jest najbardziej uczciwe i rzetelne (wynika to z faktu, iż poszczególne techniki mają w większości złożoność wielomianową, a metoda referencyjna – wykładniczą).

5.1. System Hippo

System *Hippo* [38] to zbiór narzędzi i metod, wspomagających proces analizy oraz dekompozycji systemów współbieżnych (w tym systemów cyber-fizycznych), aktywnie rozwijany pod kierunkiem prof. dr hab. inż. R. Wiśniewskiego w Uniwersytecie Zielonogórskim od 2005 roku. System jest dostępny w postaci serwisu internetowego (*www.hippo.uz.zgora.pl*), w ramach którego dostępna jest również baza testów (benchmarków). Jest to zbiór sieci Petriego, opisujących zarówno rzeczywiste systemy współbieżne (w tym systemy cyber-fizyczne), zebrane z całego świata. Ponadto, baza ta jest bardzo zróżnicowana, znajdują się w niej sieci posiadające różne właściwości (np. sieci ograniczone, sieci nieograniczone, sieci żywe, sieci, które nie są żywe itp.), w związku z czym stanowi ona świetne źródło benchmarków umożliwiających rzetelną weryfikację technik przedstawionych w niniejszej rozprawie.

5.2. Implementacja zaproponowanego rozwiązania

Zaproponowana metoda analizy żywotności części sterującej systemu cyber-fizycznego została w całości zaimplementowana przez autora rozprawy (podobnie jak metoda referencyjna). Implementacja została zrealizowana w ramach rozwijanego przez naukowców oraz studentów Uniwersytetu Zielonogórskiego systemu *Hippo*. Warto dodać, że zarówno zaproponowana metoda, jak i poszczególne techniki wchodzące w jej skład zostały zaimplementowane jako osobne moduły systemu *Hippo* i mogą

z powodzeniem znaleźć zastosowanie w dalszych badaniach prowadzonych z zastosowaniem tego systemu.

Zaproponowane rozwiązanie zostało zaimplementowane z wykorzystaniem języka programowania C++ (podobnie, jak reszta modułów dostępnych w systemie *Hippo*). W procesie implementacji wykorzystano tzw. *dobre praktyki* [105]. Kompletna metoda oraz wchodzące w jej skład techniki zostały podzielone na szereg mniejszych funkcji. Każda funkcja została sprawdzona z wykorzystaniem testów jednostkowych, aby potwierdzić poprawność implementacji oraz wyeliminować potencjalne błędy na wczesnym etapie prac programistycznych. Zarówno całe rozwiązanie, jak i poszczególne techniki zostały również zweryfikowane przed rozpoczęciem badań eksperymentalnych z wykorzystaniem wybranych benchmarków z bazy testowej *Hippo*. Do weryfikacji przygotowano także specjalnie spreparowane sieci Petriego (nie będące częścią systemu *Hippo*), w których zawarto potencjalne problematyczne schematy połączeń pomiędzy miejscami oraz tranzycjami. Jako środowisko programistyczne wykorzystano narzędzie *CLion* firmy JetBrains (dostęp do licencji uzyskano w ramach programu edukacyjnego). Wybrane fragmenty kodów źródłowych zamieszczono w Dodatku B.

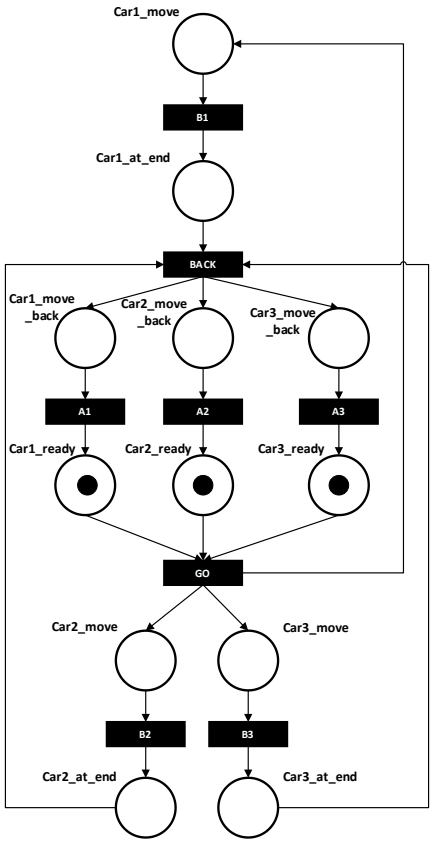
5.3. Format PNH

Format, który wykorzystują sieci testowe z bazy *Hippo* to *PNH* (ang. *Petri Net Hippo*). Format ten był podstawą opracowanej metody oraz poszczególnych technik. Głównym elementem formatu jest macierz incydencji, która odzwierciedla połączenia pomiędzy miejscami a tranzycjami. Przykładowa sieć Petriego w formie graficznej oraz opisana w formacie PNH znajduje się na rysunku 5.1, wraz z opisem elementów, na które składa się ten format.

Drugim elementem niezbędnym do opisu sieci z wykorzystaniem formatu PNH jest znakowanie początkowe sieci (przechowywany jako dodatkowy wierz w głównej macierzy incydencji). Kolumny w macierzy incydencji reprezentują miejsca sieci, a wiersze tranzycje. W macierzy incydencji można odnaleźć takie wartości jak: „**0**", „**1**" oraz „**x**". Wartość „**1**" oznacza wyjście z tranzycji a zarazem wejście do miejsca. Wartości „**x**" oznacza wyjście z miejsca a zarazem wejście do tranzycji, a wartość „**0**" oznacza brak połączenia pomiędzy miejscem a tranzycją.

Opcjonalnie, istnieje możliwość dodania informacji o sygnałach sterujących (w przypadku proponowanego rozwiązania w analizie żywotności sygnały sterujące nie są wykorzystywane). Sygnały sterujące są wykorzystywane w procesie konwersji modelu do

języka zrozumiałego przez część sterującą np. Verilog, C itp.). Format PNH umożliwia również przechowywanie nazw miejsc oraz tranzycji.



12 9	**Informacje niezbędne do wczytania pliku (liczba kolumn oraz wierszy w macierzy)**
111xxx000000 x00000010000 00010000x000 0x0000100000 00x000000001 0000010000x0 000010000x00 000000xx111x	**Macierz incydencji**
000111000000	**Wiersz ze znakowaniem początkowym**
;Places=Car1_move;Car2_move;Car3_move;Car1_ready;Car2_ready;Car3_ready;Car2_at_end;Car1_at_end;Car1_move_back; Car2_move_back;Car3_move_back;Car3_at_end ;Transitions=GO;B1;A1;B2;B3;A3;A2;BACK	**Nazwy miejsc oraz tranzycji**
;Benchmark: _3carros ;Author: University of Lisboa ;Date of creation: Mon Jun 24 03:22:43 2013 ;Last modification: 02.07.2019 ;Modified by: Marcin Wojnakowski ;Converted via: pnml2pnh (daniel.m.kur@gmail.com)	**Informacje opcjonalne**

Rysunek 5.1 Przykładowa sieć Petriego wraz z odpowiadającej jej opisem w formacie PNH

5.4. Wyniki przeprowadzonych badań eksperymentalnych

Jako platformę sprzętową, na której uruchamiano implementację opracowanych metod wykorzystano dedykowany serwer obliczeniowy o specyfikacji Intel® Xeon® Gold 5220 @2.2 GHz wyposażony w 128 GB pamięci RAM. Ponadto dokonano również weryfikacji metody na komputerze biurowym o specyfikacji Intel® Core® i7-1165G7@2.8 GHz oraz 32 GB RAM. Dokonanie dodatkowej weryfikacji (na komputerze biurowym) miało na celu sprawdzenie czy potencjalny projektant dysponujący tylko standardowym komputerem przenośnym będzie także w stanie przeprowadzić analizę żywotności z zastosowaniem proponowanego rozwiązania. Niemniej jednak, weryfikację efektywności oraz skuteczności opracowanego rozwiązania przeprowadzono na podstawie rezultatów uzyskanych w ramach testów przeprowadzonych z zastosowaniem dedykowanego serwera obliczeniowego.

Badania eksperymentalne przeprowadzono dla proponowanej metody badania żywotności omówionej w rozdziale 5 (z uwzględnieniem wszystkich technik), którą porównano do metody referencyjnej (przedstawionej w rozdziale 4). Należy w tym

miejscu podkreślić, że obie metody zostały zaimplementowane przez autora w systemie *Hippo*, wobec czego stanowią miarodajną i wiarygodną podstawę porównawczą do oszacowania efektywności i skuteczności proponowanego rozwiązania (m.in. ten sam język programowania, ten sam styl kodowania, itp.).

W tabeli 1 przedstawiono wybrane wyniki eksperymentów przeprowadzonego na serwerze obliczeniowym (pełne zestawienie znajduje się w dodatku A). Tabela 1 składa się z sześciu głównych kolumn. W pierwszej kolumnie znajduje się nazwa testu (nazwa sieci Petriego z bazy *Hippo*), w drugiej kolumnie umieszczono liczbę miejsc analizowanej sieci Petriego (wartość w nawiasie to liczba miejsc po redukcji, która jest częścią proponowanego rozwiązania), w trzeciej kolumnie zawarto liczbę tranzycji sieci (analogicznie wartość w nawiasie to liczba tranzycji po redukcji). W kolumnie czwartej znajduje się liczba stanów osiągalnych ze znakowania początkowego (wartość w nawiasie to liczba stanów dla sieci po redukcji). W tej kolumnie może również wystąpić symbol „–". Wystąpienie tego symbolu może oznaczać dwa przypadki: pierwszy, że nieudało się w wyznaczonym czasie wygenerować grafu osiągalności stanów (co wiąże się z brakiem informacji o ilości stanów w badanej sieci – ten przypadek występuje dla metody bazowej); drugi, że weryfikację żywotności zakończono przed wygenerowaniem grafu osiągalności stanów (taka sytuacja może wystąpić jedynie w proponowanym rozwiązaniu). Piąta kolumna zawiera wynik eksperymentu dla metody referencyjnej (bazowej) oraz czas obliczeń. Wynikiem metody może być: „Tak", co oznacza, że badana sieć jest żywa; symbol „–" oznacza, że nie uzyskano wyniku w założonym czasie; „Nieograniczona" oznacza, że sieć jest nieograniczona; „Nie" – oznacza, że analizowana sieć nie jest żywa. Czas podany przy wyniku to średnia arytmetyczna z 10 prób. W eksperymencie jako czas graniczny oczekiwania na wynik przyjęto 24h (dwadzieścia cztery godziny). Szósta kolumna zawiera wyniki eksperymentu dla proponowanego rozwiązania. W tej kolumnie mogą być takie wyniki jak: „Tak" oznacza, że sieć jest żywa, „Nie" oznacza, że sieć nie jest żywa, „Nieograniczona" oznacza, że sieć jest nieograniczona, „Brak silnej spójności" oznacza, że badana sieć nie jest silnie spójna (czyli jest nieograniczona i/lub nie jest żywa). Podobnie jak w przypadku metody referencyjnej czas podany przy wyniku to średnia z 10 wykonanych prób.

Tabela 1. Porównanie wyników proponowanego rozwiązania z metodą referencyjną

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)	Liczba tranzycji (po redukcji)	Liczba stanów (po redukcji)	Metoda bazowa		Proponowane rozwiązanie	
				Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
3arros	12 (2)	8 (2)	16 (2)	Tak	0,497	Tak	1,342
airplane_flight_procedure	28 (16)	17 (13)	248 (244)	Tak	51,684	Tak	37,225
brenner1	16 (10)	12 (6)	2808 (61)	Tak	5045,157	Tak	5,572
cn_crr10	80 (2)	21 (2)	- (2)	-	-	Tak	49,146
cn_crr15	120 (2)	31 (2)	- (2)	-	-	Tak	167,029
cn_crr25	200 (2)	51 (2)	- (2)	-	-	Tak	988,549
consumerReachability	5 (3)	4 (3)	3 (-)	Nieograniczona	0,016	Brak silnej spójności	0,048
effectiveness_AM_v3_corrected	46 (17)	44 (10)	2755 (82)	Tak	7017,504	Tak	84,932
lnet_p7n1	41 (2)	32 (2)	3465 (2)	Tak	10217	Tak	19,302
manufacturing_AM_deadlock	75 (10)	73 (11)	11144 (6)	Nieograniczona	57240,91	Nieograniczona	269,537
photovoltaics_single_module	139 (41)	162 (26)	- (6562)	-	-	Tak	52231,7
procesAM	29 (14)	25 (10)	74229 (-)	Nieograniczona	2977195	Nie	0,768
SHA_decoder	200 (2)	170 (2)	- (2)	-	-	Tak	4359,44
lnet_p8n1	51 (4)	40 (3)	1732 (-)	Nie	3445,46	Nie	19,798
RSA_err	43 (32)	46 (35)	- (-)	-	-	Nie	19,975

Z przeprowadzonych badań wynika, że w przypadku metody referencyjnej w przeciągu 24 godzin nie udało się uzyskać rozwiązania dla 9 badanych sieci. Oznacza to, że dla tych 9 testów metoda referencyjna nie była w stanie określić żywotności w założonym czasie. Dla porównania, metoda proponowana była w stanie przeprowadzić kompletną analizę żywotności **dla wszystkich badanych 246 sieci**.

Uśredniony czas oczekiwania na wynik w przypadku metody referencyjnej wyniósł *16901,82* ms. Natomiast dla metody proponowanej czas ten wynosił *1206,49* ms. Oznacza to, że średnio metoda proponowana okazała się **14 razy szybsza, niż metoda referencyjna**.

Przechodząc do szczegółowej analizy uzyskanych wyników warto zauważyć, że dla niektórych sieci, takich jak np. *cn_crr10, cn_crr15, cn_crr25* w przypadku metody podstawowej 24 godziny były niewystarczające do uzyskania wyniku, podczas gdy wykorzystując proponowanie rozwiązanie udało się uzyskać wynik w czasie poniżej 1 sekundy.

Kolejnym bardzo ciekawym przykładem jest sieć *procesAM*. W przypadku tej sieci, za pomocą metody referencyjnej uzyskano wynik **po upływie blisko 50 minut**. Ponadto, wynikiem tej metody była informacja, że sieć jest nieograniczona, w związku z czym metoda referencyjna nie była w stanie jednoznacznie określić żywotności sieci. Natomiast w przypadku proponowanego rozwiązania, czas obliczeń wyniósł **poniżej jednej milisekundy**. Co niezwykle istotne, proponowana metoda wprost określiła, że sieć nie jest żywa. Wynika to z faktu, że wykryta została sekwencja już na pierwszym etapie proponowanego rozwiązania. **Należy w tym miejscu podkreślić nie tylko efektywność, ale także skuteczność proponowanej metody, która była w stanie jednoznacznie określić brak żywotności sieci, podczas gdy metoda referencyjna nie była w stanie tego zweryfikować**.

Proponowane rozwiązanie posiada również pewne ograniczenia. Przykładowo, w przypadku sieci *consumerReachability* wynikiem działania proponowanej metody jest informacja o tym, że badana sieć może być nieograniczona lub może nie być żywa (brak silnej spójności). W tym przypadku nie można rozstrzygnąć właściwości żywotności sieci, ponieważ wynik jest uzyskiwany jeszcze przed rozpoczęciem procedury generowania grafu osiągalności. Z drugiej strony, metoda referencyjna również nie jest w stanie potwierdzić, czy ta sieć jest żywa. Wynika to z faktu, że *consumerReachability* jest siecią nieograniczoną. To z kolei potwierdza słuszność kierunku przyjętego przez autora

odnośnie przerywania analizy żywotności już na etapie weryfikacji silnej spójności (szerzej opisanej w rozdziale 4).

Ponadto, należy również pamiętać, że generowanie grafu osiągalności dla zredukowanej sieci ma wciąż złożoność wykładniczą. Niemniej jednak, wcześniejsze etapy (które są wielomianowe), a także zastosowane techniki redukcji pozwalają w znacznym stopniu usprawnić ten etap i tym samym znacząco skrócić czas analizy, czego wymiernym potwierdzeniem jest fakt, iż wszystkie analizowane sieci zostały z powodzeniem przeanalizowane z zastosowaniem proponowanej metody.

Reasumując należy podkreślić, że proponowana metoda była w stanie obliczyć rozwiązanie dla wszystkich 246 analizowanych sieci. Co istotne, wszystkie wyniki są w pełni zgodne z metodą referencyjną (dla testów, dla których metoda referencyjna nie była w stanie ukończyć obliczeń, przeprowadzono manualną analizę żywotności). Uwzględniając fakt, że metoda proponowana była w stanie wyznaczyć wynik w założonym czasie dla wszystkich sieci, należy stwierdzić, że **wyniki eksperymentalne potwierdziły efektywność oraz skuteczność proponowanej metody**.

5.5. Podsumowanie

W rozdziale opisano eksperymentalną weryfikację zaproponowanego rozwiązania. Wyniki zostały porównane z metodą referencyjną (szczegółowo opisaną w rozdziale 3). W rozdziale dokonano analizy wyników, które **potwierdzają efektywność oraz skuteczność metody**. Średni czas oczekiwania na wynik w przypadku metody proponowanej jest o 14 razy krótszy w porównaniu do metody referencyjnej.

Przedstawiono również sposób implementacji zaproponowanej metody. Podkreślono, że implementacja została zrealizowana przez autora rozprawy w ramach systemu *Hippo*. W rozdziale opisano także zastosowany format sieci (PNH), w którym opisane były analizowane benchmarki (sieci Petriego).

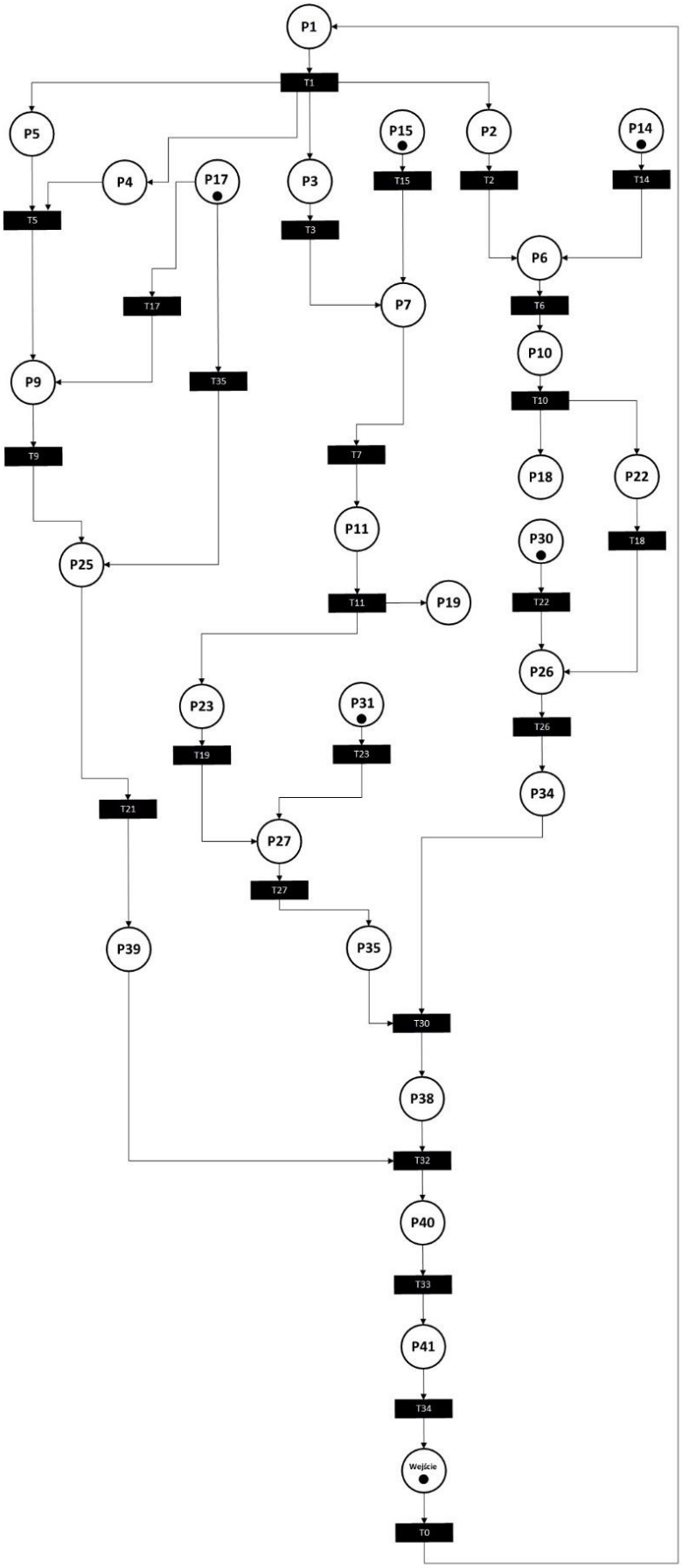
6. Wybrane przykłady ilustrujące zastosowanie opracowanej metody

W tym rozdziale zostały przedstawione wybrane przykłady zastosowania zaproponowanej metody w procesie analizy części sterującej systemu cyber-fizycznego opisanej z wykorzystaniem sieci Petriego. Wybrane przykłady ukazują atuty oraz ograniczenia proponowanej metody.

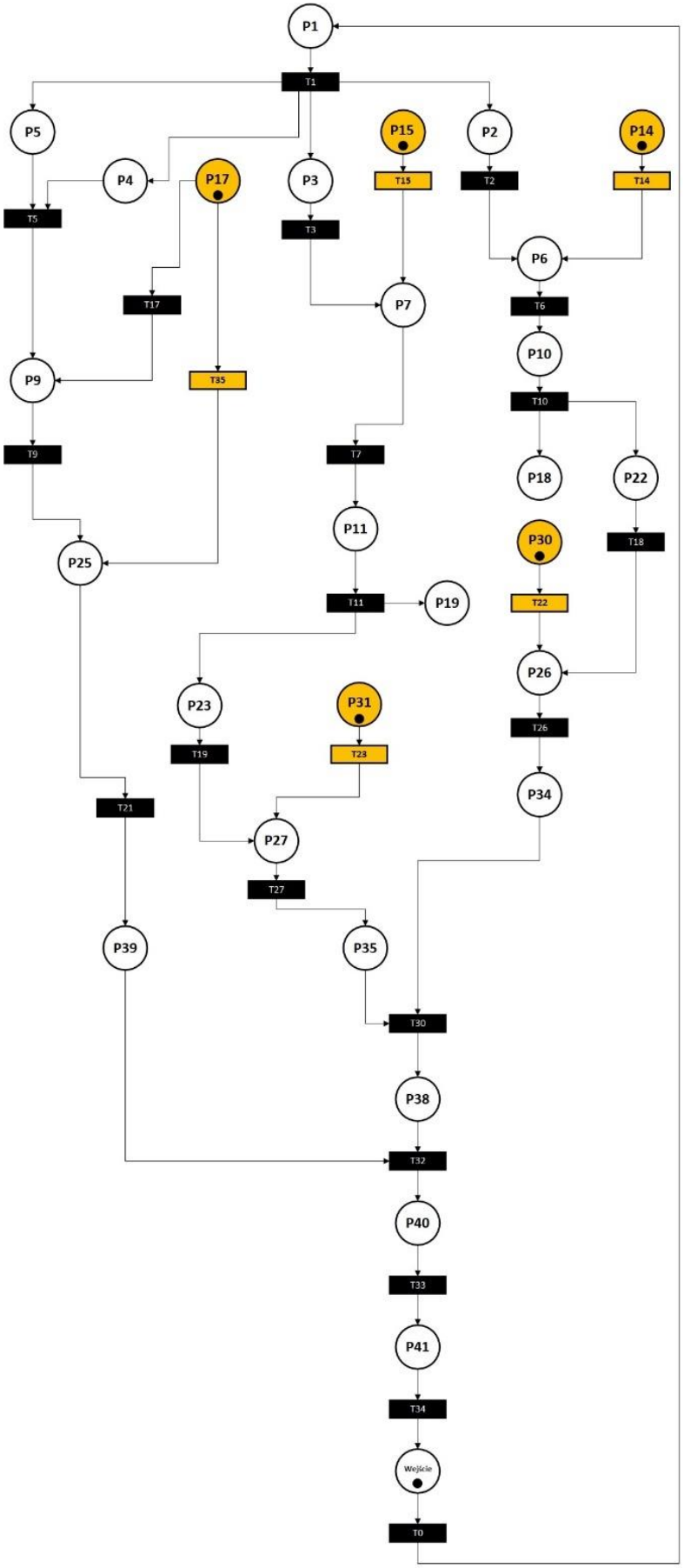
6.1. Przykład 1: sieć *procesAM*

Pierwszym zaprezentowanym przykładem jest sieć o nazwie *procesAM* z bazy *Hippo* (rysunek 6.1). Opisywana sieć składa się z 29 miejsc oraz 25 tranzycji. W przypadku tej sieci za pomocą metody referencyjnej nie udało się zweryfikować właściwości żywotności. Dla analizowanej sieci w trakcie procesu generowania grafu osiągalności okazało się, że w sieci występują miejsca nieograniczone co uniemożliwia wygenerowanie grafu osiągalności (graf osiągalności w tym przypadku byłby nieskończony). W momencie wykrycia miejsca nieograniczonego, procedurę generowania grafu osiągalności stanów zastąpiono procedurą generowania grafu pokrycia (stany wygenerowane do momentu wykrycia miejsca nieograniczonego wykorzystano w dalszej procedurze generowania grafu pokrycia). Wygenerowany graf pokrycia, który został wykorzystany w procesie weryfikacji ograniczoności, składał się z 74 229 stanów. Generowanie grafu pokrycia dla tej sieci trwało prawie 50 minut.

Zaproponowana metoda umożliwia weryfikację żywotności tej sieci. W tym przypadku wynik osiągnięto już w pierwszym etapie proponowane metody. W opisywanej sieci występują sekwencje, które uniemożliwiają, aby sieć była żywa. Czas potwierdzenia wystąpienia sekwencji trwał 0,768 ms. Na rysunku 6.2 oznaczono odróżniającym się kolorem, gdzie znajdują się sekwencje z pierwszej grupy (**S1**). W tym przypadku proponowana metoda umożliwiła weryfikację żywotności i to w bardzo krótkim czasie. W przypadku metody referencyjnej wynikiem osiągniętym po prawie 50 minutach nie była informacja o żywotności, ale o braku ograniczoności badanej sieci.



Rysunek 6.1. Sieć Petriego *procesAM*



Rysunek 6.2. Sekwencje odnalezione w sieci *procesAM*

6.2. Przykład 2: sieci z rodziny cn_*crr*

Drugim prezentowanym przykładem jest sieć, a dokładniej grupa (rodzina) sieci *cn_crr* (w bazie *Hippo* znajdują się cztery sieci z tej grupy). W przypadku tych sieci, dzięki etapowi redukcji iteracyjnej z proponowanego rozwiązania, zmniejszono liczbę stanów w grafie osiągalności co pozwoliło na weryfikację żywotności. W przypadku metody referencyjnej dla trzech z czterech sieci nie udało się wygenerować grafu w ciągu 24 godzin. Największa z rodziny sieć *cn_crr25* składa się z dwustu miejsc oraz pięćdziesięciu jeden tranzycji. Po zrealizowaniu redukcji, sieć zmniejszyła się do dwóch miejsc oraz dwóch tranzycji. Graf osiągalności dla zredukowanej sieci liczył dwa stany, co umożliwiło bez żadnych przeszkód przeprowadzić weryfikację żywotności. Proponowane rozwiązanie zwróciło wynik po 988,549 ms (sieć jest żywa), a najbardziej czasochłonny okazał się tu proces redukcji sieci.

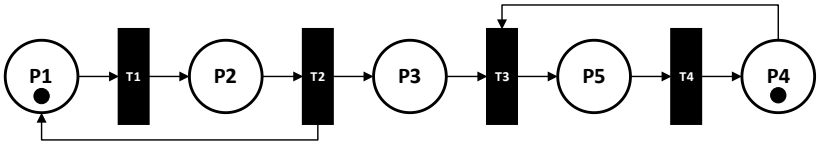
Dla najmniejszej sieci z tej rodziny (*cn_crr7*) metoda referencyjna umożliwiła uzyskanie wyniku w czasie 4 429,724 ms. Graf osiągalności stanów, który został wykorzystany w metodzie referencyjnej liczył 2187 stanów. Graf osiągalności w proponowanym rozwiązaniu liczył 2 stany (dla sieci po redukcji), czas oczekiwania na wynik wyniósł 64,638 ms. Jak widać w przypadku tych sieci proponowane rozwiązanie ma przewagę nad metodą referencyjną.

Istnieją jednak sieci w bazie testów *Hippo*, które uwypuklają ograniczenia proponowanego rozwiązania, choć trzeba także stwierdzić, że w metodzie referencyjnej również występują te ograniczenia i to w jeszcze większej skali. Wspominanym ograniczeniem jest brak możliwości uzyskania jednoznacznego wyniku (sieć żywa lub sieć nie jest żywa) dla sieci, które nie są ograniczone. Przykład takiej sieci pokazano w kolejnym podrozdziale.

6.3. Przykład 3: sieć *consumerReachability*

Sieć *consumerReachability* z bazy *Hippo* (rysunek 6.3) składa się z 5 miejsc oraz 4 tranzycji. Metoda referencyjna potwierdziła brak ograniczoności tej sieci w czasie 0,016 ms. Metoda referencyjna nie jest wstanie w tym przypadku rozstrzygnąć, czy sieć jest żywa. W przypadku metody proponowanej również nie jest możliwe rozstrzygnięcie czy sieć jest żywa. Metoda proponowana zwróciła informację, że sieć nie jest silnie spójna, co oznacza, że sieć może być nieograniczona lub nie jest żywa, a to oznacza, że dalsze etapy proponowanego rozwiązania nie będą skuteczne. Wynik z informacją, że sieć nie jest silnie spójna proponowane rozwiązanie zwróciło po 0,048 ms. Można zauważyć, że czas

metody proponowanej w tym przypadku jest gorszy o 0,032 ms. Jest to związane z etapem poszukiwania sekwencji, który w metodzie referencyjnej nie występuje oraz niewielkiej liczbie stanów, po analizie których wykryto nieograniczoność sieci.



Rysunek 6.3. Sieć nieograniczona *consumerReachability*

6.4. Przykład 4: sieci z rodziny *effectiveness_AM_unbounded*

Inną siecią z bazy *Hippo*, dla której występuje omawiane wcześniej ograniczenie jest autorski przykład *effectiveness_AM_v1_unbounded*. Dla tej sieci metoda referencyjna zwróciła wynik, że sieć nie jest ograniczona. Aby tego dokonać wyznaczono 4 676 stanów w czasie 7 546,927 ms. W przypadku metody proponowanej informację o tym, że sieć może być nieograniczona lub nie jest żywa otrzymano w czasie 63,16 ms. Jest to czas krótszy od metody referencyjnej o 7 483,767 ms. Metoda proponowana nie musiała wyznaczać grafu osiągalności, bo już na etapie weryfikacji silnej spójności przerwano działanie algorytmu i uzyskano wynik.

Bardzo ciekawym przykładem jest także sieć *effectiveness_AM_v2_unsafe*. W przypadku metody referencyjnej dla tej sieci nie uzyskano wyniku w ciągu 24h, a proponowana metoda umożliwiła weryfikację żywotności w czasie zaledwie 19,702 ms. Było to możliwe dzięki analizie sekwencji, które są całkowicie autorskim rozwiązaniem. W przytoczonej sieci odnaleziono sekwencję **S2**, której wystąpienie uniemożliwia, aby sieć mogła być żywa.

6.5. Przykład 5: sieci z rodziny *photovoltaics_single_module*

Ostatnim przytoczonym przykładem sieci Petriego z bazy *Hippo*, dla której proponowane rozwiązanie stanowczo wykazuje swoją przewagę jest *photovoltaics_single_module*. W przypadku tej sieci metoda referencyjna nie była wstanie zwrócić wyniku w ciągu 24h. Proponowane rozwiązanie potwierdziło, że sieć jest żywa. W tym przypadku sieć przeszła przez wszystkie etapy proponowanego rozwiązania. Na etapie poszukiwania sekwencji nie odnaleziono żadnej, więc uruchomiono kolejny etap, w którym badana była silna spójność. W tym etapie potwierdzono, że sieć jest silnie spójna i nastąpiło przejście do kolejnego etapu proponowanego rozwiązania jakim są redukcje iteracyjne. W przypadku

redukcji udało się zmniejszyć liczbę miejsc z 139 do 41, a liczba tranzycji zmniejszyła się z 162 do 26.

Po etapie redukcji przystąpiono do wyznaczania grafu osiągalności stanów. W trakcie wyznaczania grafu potwierdzono, że sieć jest ograniczona. Wyznaczony graf osiągalności stanów liczył 6 562 stany. Na podstawie tego grafu wyznaczono silnie spójne komponenty. W przypadku tej sieci wyznaczono 1 taki komponent, w którym znajdowały się wszystkie tranzycje z sieci, co potwierdza żywotność badanej sieci. Przejście przez wszystkie etapy proponowanego rozwiązania i potwierdzenie żywotności trwało 52 231,694 ms, czyli poniżej jednej minuty.

6.6. Podsumowanie

W rozdziale przedstawiono kilka przykładowych sieci dostępnych w bazie testowej *Hippo*, które potwierdzają sprawność oraz efektywność proponowanego rozwiązania. Pokazano również główne ograniczenie proponowanego rozwiązania, które również występuje w przypadku metody referencyjnej (jest to ogólna słabość metod, które wykorzystują graf osiągalności do weryfikacji żywotności). Wykazano, że proponowane rozwiązanie stara się omijać wpływ tego ograniczenia na ostateczny wynik (np. poprzez wykorzystanie poszukiwania sekwencji, które są wstanie potwierdzić, że analizowana sieć nie jest żywa; stosowanie redukcji sieci itp.).

Przytoczono również przykład sieci, dla której proponowane rozwiązanie musiało przejść przez wszystkie etapy, aby w pełni zrealizować analizę żywotności. W tym przypadku metoda referencyjna nie była wstanie uzyskać wyniku w ciągu 24h, a proponowane rozwiązanie zrealizowało to zadanie w czasie poniżej jednej minuty co potwierdza jego sprawność oraz efektywność mimo posiadanych ograniczeń.

7. Podsumowanie, wnioski oraz kierunki dalszych prac

W pracy przedstawiono problematykę weryfikacji żywotności części sterującej systemów cyber-fizycznych modelowanych z zastosowaniem sieci Petriego. Dokonano wprowadzenia do tematyki systemów cyber-fizycznych oraz określono w jakich obszarach przemysłu oraz życia codziennego można spotkać systemy tej klasy. Następnie, opisano czym są sieci Petriego i dlaczego warto je wykorzystywać do projektowania oraz weryfikacji poprawności części sterującej systemów CPS. Przedstawiono także wybrane zagadnienia z teorii grafów oraz teorii sieci Petriego, które są niezbędne do zrozumienia opracowanych technik badania żywotności.

Istotnym aspektem pracy jest przedstawiona referencyjna metoda analizy żywotności, do której porównano metodę proponowaną w rozprawie. Omówiono najważniejsze etapy tego rozwiązania, wskazując na najistotniejsze ograniczenia. Następnie, zaprezentowano autorską metodę analizy żywotności części sterującej systemu cyber-fizycznego opisanej siecią Petriego. W szczególności, proponowana metoda składa się z szeregu technik, które umożliwiają kompleksową weryfikację systemu.

Bardzo ważnym elementem rozprawy były przeprowadzone badania eksperymentalne. W tym celu autor zaimplementował (z zastosowaniem języka C++) zarówno metodę referencyjną, jak i metodę proponowaną (z uwzględnieniem wszystkich opracowanych technik), a następnie przeprowadził analizę żywotności dla 246 testowych sieci Petriego.

7.1. Potwierdzenie tezy pracy

Główny celem pracy było opracowanie efektywnej oraz skutecznej metody analizy żywotności części sterującej systemu cyber-fizycznego opisanej z zastosowaniem sieci Petriego. Teza pracy została sformułowana następująco: ***możliwe jest przeprowadzenie analizy żywotności części sterującej systemu cyber-fizycznego specyfikowanej z zastosowaniem sieci Petriego w sposób efektywny oraz skuteczny.***

W ramach niniejszej rozprawy najpierw dokonano syntetycznego przeglądu istniejących metod żywotności sieci Petriego. Wskazano, że dostępne koncepcje przede wszystkim bazują na budowie oraz późniejszej analizie grafu osiągalności stanów systemu, a ich największym ograniczeniem jest wykładnicza złożoność obliczeniowa. Dlatego też, w ramach pracy zdecydowano się opracować **efektywną** i **skuteczną metodę analizy żywotności części sterującej system cyber-fizycznego opisanej siecią Petriego**.

Zrealizowana metoda składa się z szeregu mniejszych, komplementarnych technik będących zarówno koncepcjami autorskimi, jak i stanowiących modyfikacje istniejących metod. W tym miejscu należy podkreślić, że aż cztery etapy (z pięciu) proponowanej metody charakteryzują się **wielomianową złożonością obliczeniową**.

Opracowane techniki zostały zaimplementowane (w formie kompletnej metody) w systemie *Hippo*, z zastosowaniem języka programowania C++, co stanowiło podstawę przeprowadzonych badań eksperymentalnych. W tym celu porównano proponowane rozwiązanie do metody referencyjnej. **Wyniki przeprowadzonych badań eksperymentalnych jednoznacznie potwierdzają efektywność oraz skuteczność opracowanej metody.** W szczególności, proponowana metoda była w stanie **wyznaczyć rozwiązanie w założonym czasie dla wszystkich badanych 246 sieci testowych (potwierdzenie efektywności)**. Co istotne, dla **wszystkich przypadków uzyskano poprawne rozwiązanie (potwierdzenie skuteczności)**. Dla porównania, metoda referencyjna nie była w stanie ukończyć obliczeń dla 9 badanych sieci. Co więcej, proponowane rozwiązanie okazało się średnio aż 14 razy szybsze niż metoda referencyjna. **Podsumowując powyższe należy stwierdzić, że teza pracy została potwierdzona zarówno analitycznie (wielomianowa złożoność obliczeniowa czterech z pięciu z opracowanych technik, z których dwie mogą stanowić niezależne metody analizy żywotności), jak i eksperymentalnie (uzyskanie poprawnych wyników dla wszystkich analizowanych sieci Petriego).**

7.2. Oryginalne wyniki pracy

Poniżej zestawiono oryginalne elementy zrealizowanej pracy doktorskiej:

- Opracowano techniki poszukiwania trzech rodzajów sekwencji (wzorców), których wystąpienie w sieci Petriego jednoznacznie wskazuje, że sieć nie jest żywa. Najważniejszą zaletą opracowanych metod jest ich wielomianowa złożoność obliczeniowa. Opracowane metody zaimplementowano programowo oraz zweryfikowano eksperymentalnie.
- Opracowano oraz zaimplementowano programowo technikę badania silnej spójności sieci Petriego (metoda bazuje na istniejącym rozwiązaniu, które zaadaptowano oraz usprawniono na potrzeby pracy). Opracowana metoda ma wielomianową złożoność obliczeniową.

- Opracowano technikę iteracyjnych (cyklicznych) redukcji sieci Petriego oraz zrealizowano implementację programową wszystkich omawianych w pracy metod redukcji sieci (FSP, FST, FPP, FPT, FESP) wraz z iteracyjną techniką redukcji.
- Opracowano oraz zaimplementowano programowo technikę wyznaczania grafu pokrycia umożliwiającą przerwanie algorytmu w przypadku wykrycia miejsca nieograniczonego (jest to usprawniona klasyczna metoda wyznaczania grafu osiągalności).
- Opracowano oraz zaimplementowano programowo technikę badania żywotności sieci Petriego na podstawie wyznaczonego grafu pokrycia z zastosowaniem usprawnionego algorytmu wyszukującego komponenty silnie spójne w wyznaczonym grafie pokrycia sieci Petriego.
- Zrealizowano implementację programową kompletnej klasycznej metody analizy żywotności sieci Petriego (*metoda referencyjna*).
- Opracowano oraz zaimplementowano programowo kompletną metodę analizy żywotności sieci Petriego (*metoda proponowana*), a następnie przeprowadzono badania eksperymentalne potwierdzające skuteczność oraz sprawność opracowanej metody (w porównaniu do *metody referencyjnej*).

Badania zamieszczone w pracy wykonano w ramach projektu badawczego OPUS realizowanego w latach 2020-2024 (temat projektu: „Analiza oraz dekompozycja części sterującej systemu cyber-fizycznego opisanego z zastosowaniem interpretowanej sieci Petriego", nr projektu 2019/35/B/ST6/01683, kierownik projektu: prof. dr hab. inż. Remigiusz Wiśniewski) finansowanego ze środków Narodowego Centrum Nauki. Należy tutaj podkreślić, że autor rozprawy w latach 2020-2023 był stypendystą wspomnianego grantu, a obecnie jest członkiem zespołu badawczego, który kontynuuje badania z zakresu analizy, weryfikacji oraz dekompozycji części sterującej systemu CPS opisanej z zastosowaniem sieci Petriego.

7.3. Kierunki dalszych prac

Opracowane w pracy metody oraz algorytmy posiadają pewne ograniczenia, dlatego najważniejszymi kierunkami dalszych prac wydają się być:

- poszukiwanie kolejnych sekwencji (wzorców), które potwierdzą, że badana sieć nie jest żywa;

- zbadanie możliwości generowania grafu pokrycia (lub innej struktury alternatywnej) dla sieci nieograniczonych pod kątem dalszej analizy żywotności;
- poszukiwanie nowych technik redukcyjnych;
- opracowanie technik oraz algorytmów nakierowanych na potrzeby przemysłu w zakresie modelowania, projektowania oraz weryfikacji poprawności specyfikacji systemów cyber-fizycznych opisanych sieciami Petriego;
- na podstawie doświadczeń z przemysłem: opracowanie nowych, autorskich klas pod kątem modelowania części sterującej systemu cyber-fizycznego (tzn. specyfikacji, które nie będą opierały się o tradycyjne klasy sieci Petriego), a także przygotowanie specjalistycznych algorytmów do weryfikacji wymaganych właściwości dla nowoopracowanych klas;
- opracowanie techniki modelowania systemów cyber-fizycznych ściśle w oparciu o wymogi formalne systemu (tzn. również z uwzględnieniem sygnałów wejściowych oraz wyjściowych);
- wykorzystanie technik sztucznej inteligencji do analizy żywotności części sterującej systemów CPS.

Dodatek A. Szczegółowe wyniki eksperymentów

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)	Liczba tranzycji (po redukcji)	Liczba stanów (po redukcji)	Metoda bazowa		Proponowane rozwiązanie	
				Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
2pasber	15 (13)	18 (16)	52 (40)	Nieograniczona	1,299	Nieograniczona	3,531
3carros	12 (2)	8 (2)	16 (2)	Tak	0,497	Tak	1,342
adam1	24 (24)	12 (12)	18 (18)	Tak	0,833	Tak	5,041
agermala1	8 (7)	4 (4)	4 (4)	Tak	0,082	Tak	0,256
agostini1	7 (2)	5 (2)	5 (2)	Tak	0,171	Tak	0,202
airplane_flight_procedure	28 (16)	17 (13)	248 (244)	Tak	51,684	Tak	37,225
alfa_net_copy_milling_machine_subprocess	29 (7)	25 (7)	17 (3)	Nieograniczona	0,224	Nieograniczona	6,081
alfa_net_copy_milling_machine_subprocess	32 (2)	27 (2)	230 (2)	Tak	33,247	Tak	11,942
alfa_net_copy_milling_subprocess	30 (3)	26 (3)	43 (3)	Nie	2,365	Nie	6,734
automation3	17 (12)	16 (11)	50 (0)	Nie	2,526	Brak silnej spójności	2,994
baldlszi1	18 (18)	16 (16)	8 (0)	Nie	0,284	Brak silnej spójności	2,881
barkaoui1	11 (7)	13 (8)	19 (8)	Tak	10,645	Tak	1,948
barkaoui2	10 (9)	8 (7)	7 (6)	Tak	0,166	Tak	0,707
barkaoui3	18 (14)	13 (9)	38 (20)	Tak	1,283	Tak	2,47
basile1	21 (7)	17 (6)	120 (5)	Tak	11,761	Tak	2,894

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
bause1	11	(9)	11	(9)	14	(5)	Nie	0,362	Nie	0,768
beverage	16	(2)	13	(2)	37	(2)	Tak	1,076	Tak	1,276
beverage_production	20	(2)	16	(2)	44	(2)	Tak	1,945	Tak	2,626
beverage_production_part2	16	(2)	13	(2)	37	(2)	Tak	1,579	Tak	1,181
bit_protocol	8	(6)	7	(5)	4	(4)	Nieograniczona	0,021	Nieograniczona	0,28
board_game	13	(13)	13	(13)	3	(0)	Nieograniczona	0,016	Brak silnej spójności	0,738
bouillard1	12	(6)	12	(3)	27	(3)	Tak	0,882	Tak	0,782
brenner1	16	(10)	12	(6)	2808	(61)	Tak	5045,157	Tak	5,572
bridge	8	(2)	6	(2)	10	(2)	Tak	0,192	Tak	1,301
bridge_semaphore	9	(8)	6	(6)	9	(9)	Tak	0,146	Tak	0,488
campos1	11	(9)	8	(6)	16	(6)	Tak	0,373	Tak	11,143
campos2	12	(11)	10	(9)	8	(7)	Tak	0,196	Tak	1,058
campos3	15	(12)	11	(9)	36	(15)	Tak	1,144	Tak	2,137
chiola1	16	(15)	15	(14)	12	(11)	Tak	0,283	Tak	2,323
chrzastowski1	10	(6)	10	(5)	15	(4)	Tak	0,371	Tak	0,966

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
chrzastowski2	22	(2)	21	(2)	27	(2)	Tak	1,214	Tak	3,758
cncrr001	7	(2)	4	(2)	4	(2)	Tak	0,09	Tak	0,254
cncrr002	11	(5)	7	(4)	30	(5)	Tak	1,692	Tak	0,512
CNC_machine	7	(2)	5	(1)	6	(0)	Nie	0,108	Nie	0,014
cn_crr10	80	(2)	21	(2)	0	(2)	-	0	Tak	49,146
cn_crr15	120	(2)	31	(2)	0	(2)	-	0	Tak	167,029
cn_crr25	200	(2)	51	(2)	0	(2)	-	0	Tak	988,549
cn_crr7	56	(2)	15	(2)	2187	(2)	Tak	4429,724	Tak	64,638
coloured	4	(2)	3	(2)	4	(2)	Tak	0,077	Tak	0,123
communications_protocol	6	(2)	8	(2)	6	(2)	Tak	0,134	Tak	0,67
ConsistentExample	29	(29)	25	(25)	54	(0)	Nie	2,92	Nie	11,525
ConsistentExampleMessageView	10	(9)	6	(5)	9	(0)	Nie	0,159	Nie	0,021
*consumer*Reachability	5	(3)	4	(3)	3	(0)	Nieograniczona	0,016	Brak silnej spójności	0,048
cortadella1	9	(9)	6	(6)	7	(7)	Tak	0,129	Tak	0,305
cp2	8	(6)	6	(4)	9	(7)	Nie	0,154	Nie	0,656

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
credit_procedure	10	(2)	8	(1)	15	(0)	Nie	0,326	Nie	0,024
crossroadSM_FPGA	32	(16)	12	(8)	12	(8)	Tak	0,302	Tak	5,94
dataflow_computation	11	(11)	7	(7)	12	(0)	Nie	0,288	Nie	0,024
decision_making_system_v1	75	(11)	73	(12)	54	(0)	Nieograniczona	0,989	Brak silnej spójności	197,301
decision_making_system_v2	75	(10)	73	(11)	11144	(6)	Nieograniczona	56185,954	Nieograniczona	251,139
decision_making_system_v3	75	(5)	74	(6)	11144	(5)	Nieograniczona	55103,318	Nieograniczona	241,821
decision_making_system_v4	77	(3)	76	(4)	7204	(3)	Tak	58262,399	Tak	331,446
desel1	15	(2)	9	(2)	129	(2)	Tak	13,876	Tak	1,17
dingle1	20	(16)	20	(16)	17	(13)	Tak	0,455	Tak	3,976
dining_philosophers	15	(15)	10	(10)	12	(12)	Nieograniczona	0,126	Nieograniczona	1,143
dongen1	15	(6)	17	(7)	20	(7)	Tak	0,503	Tak	2,695
effectiveness_AM_v1_unbounded	48	(23)	45	(16)	4676	(0)	Nieograniczona	7546,927	Brak silnej spójności	63,16
effectiveness_AM_v2_unsafe	48	(9)	52	(6)	0	(0)	-	0	Nie	19,702
effectiveness_AM_v3_corrected	46	(17)	44	(10)	2755	(82)	Tak	7017,504	Tak	84,932
Elevator01	12	(10)	17	(15)	98	(78)	Nie	5,288	Nie	4,341

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
elevator_2	8	(2)	8	(2)	8	(2)	Tak	0,146	Tak	0,276
Entrance1	4	(2)	6	(2)	4	(2)	Tak	0,108	Tak	0,191
eshuis1	17	(9)	15	(7)	23	(6)	Tak	0,582	Tak	2,003
esparza1	11	(10)	10	(9)	21	(0)	Nie	0,62	Brak silnej spójności	0,363
esparza2	15	(13)	13	(11)	31	(24)	Tak	0,809	Tak	2,784
esparza3	13	(9)	15	(10)	35	(6)	Nie	1,218	Nie	1,327
ExampleStateView	13	(12)	10	(9)	48	(0)	Nie	1,697	Nie	0,423
Exe5_split	8	(2)	6	(2)	8	(2)	Tak	0,149	Tak	0,225
exOR	10	(8)	7	(5)	2	(2)	Nie	0,06	Nie	0,48
fernandez1	13	(5)	11	(5)	10	(4)	Tak	0,195	Tak	0,888
fernandez2	11	(5)	10	(5)	9	(4)	Tak	0,179	Tak	0,583
fernandez3	21	(10)	20	(11)	20	(8)	Tak	0,562	Tak	4,202
fig311_01	6	(6)	6	(6)	9	(4)	Tak	0,181	Tak	0,374
FMS_main_SIPN	7	(2)	4	(2)	4	(2)	Tak	0,11	Tak	0,192
four_philosophers	12	(12)	8	(8)	7	(7)	Tak	0,241	Tak	0,561

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
frame_manufact	13	(2)	10	(2)	17	(2)	Tak	0,362	Tak	0,597
franczok1	10	(3)	14	(4)	10	(3)	Tak	0,212	Tak	0,633
gals-example	6	(6)	4	(4)	4	(4)	Tak	0,077	Tak	0,184
gaubert1	16	(7)	8	(4)	8	(4)	Tak	0,171	Tak	0,93
gaubert2	8	(7)	4	(4)	4	(4)	Tak	0,083	Tak	0,257
girault1	12	(12)	6	(6)	8	(8)	Tak	0,165	Tak	7,941
girault2	16	(12)	10	(6)	20	(8)	Tak	0,47	Tak	1,107
girault3	14	(13)	9	(8)	12	(10)	Tak	0,301	Tak	0,886
girault4	10	(10)	7	(7)	20	(20)	Tak	0,444	Tak	2,262
girault7	11	(5)	7	(4)	7	(3)	Tak	0,164	Tak	0,406
girault8	8	(8)	8	(8)	16	(16)	Tak	0,449	Tak	0,504
hack1	13	(11)	13	(11)	44	(27)	Tak	1,764	Tak	1,632
HAN	24	(21)	40	(36)	1300	(1297)	Tak	3622,669	Tak	1770,086
handling_of_incoming_order	17	(10)	14	(7)	23	(0)	Nie	0,594	Nie	0,177
hard_case	9	(2)	3	(2)	3	(2)	Tak	0,067	Tak	0,404

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
health_care_process	6	(6)	8	(8)	6	(0)	Nie	0,14	Nie	0,039
hee1	14	(13)	12	(11)	20	(19)	Tak	0,525	Tak	2,053
hee2	13	(12)	11	(10)	12	(11)	Tak	0,251	Tak	1,037
heiner1	22	(22)	20	(20)	118	(118)	Tak	10,745	Tak	10,222
holliday1	20	(15)	15	(10)	152	(11)	Tak	24,798	Tak	2,267
holloway1	13	(5)	10	(3)	28	(3)	Tak	0,783	Tak	0,873
hulgaard1	19	(18)	12	(12)	13	(13)	Tak	0,305	Tak	2,535
IEC	15	(2)	12	(2)	24	(2)	Tak	0,534	Tak	1,135
img_280	8	(2)	6	(2)	11	(2)	Tak	0,247	Tak	0,264
invariants_exponent_3_2	6	(2)	2	(2)	2	(2)	Tak	0,06	Tak	0,309
invariants_exponent_3_3	9	(2)	3	(2)	3	(2)	Tak	0,069	Tak	0,234
invariants_exponent_3_4	12	(2)	4	(2)	4	(2)	Tak	0,077	Tak	0,714
inv_exp_20_layers	200	(7)	20	(3)	1	(1)	Nie	0,057	Nie	459,272
inv_exp_4_layers	200	(6)	4	(3)	1	(1)	Nie	0,167	Nie	76,924
inv_exp_4_layers_v2	200	(5)	4	(3)	1	(1)	Nie	0,052	Nie	63,388

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
jeng1	14	(6)	16	(7)	25	(5)	Tak	0,691	Tak	1,252
jeng2	10	(3)	9	(4)	8	(3)	Tak	0,163	Tak	1,123
julvez_1	11	(2)	8	(2)	48	(2)	Tak	1,675	Tak	0,466
julvez_2	12	(5)	11	(5)	10	(4)	Tak	0,207	Tak	2,24
kavi1	18	(11)	20	(13)	180	(39)	Tak	25,025	Tak	5,247
kovalyov92	4	(4)	6	(6)	4	(4)	Tak	0,142	Tak	0,146
lab5	8	(2)	6	(2)	8	(2)	Tak	0,209	Tak	0,298
lasire1	15	(3)	13	(3)	13	(2)	Tak	0,261	Tak	1,052
li1	17	(2)	17	(2)	32	(2)	Tak	2,188	Tak	3,531
li2	12	(3)	12	(3)	18	(2)	Tak	0,424	Tak	1,671
li3	11	(5)	14	(6)	14	(4)	Tak	0,403	Tak	0,769
lnet_p10n1	12	(2)	9	(2)	38	(2)	Tak	2,221	Tak	0,519
lnet_p1n1	6	(2)	4	(2)	4	(2)	Tak	0,085	Tak	0,184
lnet_p1n2	6	(3)	5	(3)	4	(2)	Tak	0,255	Tak	0,18
lnet_p1n4	9	(2)	6	(2)	8	(2)	Tak	0,151	Tak	0,358

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
lnet_p2n2	13	(2)	7	(2)	7	(2)	Tak	0,162	Tak	0,605
lnet_p2n3	9	(2)	7	(2)	7	(2)	Tak	0,139	Tak	0,314
lnet_p4n1	37	(3)	41	(4)	37	(3)	Tak	3,033	Tak	27,674
lnet_p5n1	11	(2)	9	(2)	18	(2)	Tak	0,488	Tak	0,71
lnet_p5n2	13	(2)	11	(2)	27	(2)	Tak	0,736	Tak	0,717
lnet_p5n3	10	(2)	8	(2)	13	(2)	Tak	0,746	Tak	0,47
lnet_p6n1	12	(3)	11	(3)	21	(2)	Tak	0,534	Tak	0,65
lnet_p6n2	14	(7)	16	(9)	21	(5)	Nieograniczona	0,188	Nieograniczona	1,496
lnet_p6n3	6	(3)	5	(3)	4	(2)	Tak	0,078	Tak	0,183
lnet_p7n1	41	(2)	32	(2)	3465	(2)	Tak	10217,002	Tak	19,302
lnet_p8n1	51	(4)	40	(3)	1732	(0)	Nie	3445,46	Nie	19,798
lnet_p8n2	28	(2)	17	(2)	81	(2)	Tak	4,46	Tak	5,121
lnet_p8n3	21	(2)	17	(2)	81	(2)	Tak	4,168	Tak	2,377
lnet_p8n4	15	(10)	16	(11)	56	(17)	Tak	2,834	Tak	2,455
lnet_p9n1	14	(9)	16	(11)	38	(9)	Tak	1,419	Tak	2,338

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
machine_shop	12	(11)	9	(8)	1	(1)	Nie	0,037	Nie	1,096
manufacturing_AM_deadlock	75	(10)	73	(11)	11144	(6)	Nieograniczona	57240,909	Nieograniczona	269,537
manufacturing_AM_v1_initial	55	(5)	61	(6)	687	(5)	Nieograniczona	198,549	Nieograniczona	102,314
manufacturing_AM_v2_exp	55	(8)	60	(9)	687	(6)	Nieograniczona	196,458	Nieograniczona	127,588
manufacturing_AM_v3_final	57	(3)	63	(4)	486	(3)	Tak	167,779	Tak	102,063
maruster1	13	(5)	14	(4)	22	(3)	Tak	0,674	Tak	2,386
maruster2	20	(2)	14	(2)	46	(2)	Tak	1,899	Tak	24,937
maruster3	12	(3)	13	(4)	13	(3)	Tak	0,942	Tak	1,189
maruster4	16	(2)	13	(2)	66	(2)	Tak	2,84	Tak	1,155
medeiros1	9	(6)	13	(9)	8	(5)	Tak	0,22	Tak	0,644
medeiros2	10	(7)	8	(5)	9	(5)	Tak	0,243	Tak	0,419
miczulski1	9	(4)	8	(4)	9	(3)	Tak	0,194	Tak	0,971
miling_machine	20	(2)	16	(2)	69	(2)	Tak	3,373	Tak	2,166
milling	20	(2)	16	(2)	69	(2)	Tak	3,266	Tak	2,281
mixer	19	(2)	15	(2)	43	(2)	Tak	1,591	Tak	2,129

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
mixer_mod1	20	(2)	16	(2)	44	(2)	Tak	1,745	Tak	2,942
mixer_mod2	8	(2)	5	(2)	6	(2)	Tak	0,11	Tak	0,325
mixer_one_cup	16	(2)	13	(2)	37	(2)	Tak	1,139	Tak	1,291
multi_robot	9	(7)	6	(4)	12	(4)	Tak	0,247	Tak	0,3
mutual_exclusion	9	(8)	7	(6)	10	(9)	Tak	0,18	Tak	0,439
net1	6	(3)	5	(3)	7	(0)	Nie	0,15	Nie	0,013
net2	7	(7)	6	(6)	9	(9)	Nieograniczona	0,051	Nieograniczona	0,214
net3	8	(4)	8	(4)	10	(3)	Tak	0,191	Tak	0,444
net4	8	(4)	7	(4)	6	(3)	Tak	0,126	Tak	2,693
np3	6	(6)	3	(3)	3	(3)	Tak	0,068	Tak	0,352
np5	10	(10)	5	(5)	5	(5)	Tak	0,095	Tak	0,657
oil_separator_cover_alfa_net	27	(2)	21	(2)	35	(2)	Tak	1,165	Tak	13,184
oil_separator_cover_s_net	29	(7)	25	(7)	41	(5)	Tak	1,617	Tak	6,23
oil_separator_cover_s_net_v2	15	(7)	11	(7)	9	(5)	Tak	0,19	Tak	1,408
oil_separator_cover_s_net_v3	11	(7)	11	(7)	9	(5)	Tak	0,204	Tak	0,759

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
oneWayTransmissionSystem	9	(2)	6	(2)	12	(2)	Tak	0,228	Tak	0,319
parallel_managing_automatic_devices	11	(7)	7	(5)	11	(6)	Nieograniczona	0,089	Nieograniczona	0,854
parking_lot	6	(5)	8	(7)	8	(3)	Tak	0,247	Tak	0,342
pascal_reach	10	(8)	8	(6)	10	(6)	Tak	0,205	Tak	0,491
pcncp	6	(6)	4	(4)	4	(4)	Tak	0,105	Tak	0,803
philosophers_2	14	(12)	10	(8)	22	(18)	Tak	0,556	Tak	1,159
philosophers_2_rev_2	14	(12)	10	(8)	22	(18)	Tak	0,593	Tak	1,149
philosophers_5	20	(20)	15	(15)	82	(82)	Tak	6,86	Tak	9,943
photovoltaics_single_module	139	(41)	162	(26)	0	(6562)	-	0	Tak	52231,69
ping1	12	(7)	14	(8)	20	(8)	Tak	0,608	Tak	0,949
pnbrexpl_02	13	(2)	10	(2)	21	(2)	Tak	0,611	Tak	1,348
pnbrexpl_03	9	(4)	8	(4)	9	(3)	Tak	0,24	Tak	0,363
pnbrexpl_04	13	(8)	13	(8)	21	(6)	Tak	0,546	Tak	0,986
pnbrexpl_05	7	(2)	6	(2)	7	(2)	Tak	0,127	Tak	0,204
pnbrexpl_06	11	(2)	9	(2)	11	(2)	Tak	0,379	Tak	0,497

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
pnbrexpl_07	11	(2)	12	(2)	29	(2)	Tak	1,171	Tak	4,914
pnbrexpl_08	13	(4)	11	(4)	15	(3)	Tak	0,303	Tak	0,715
pnbrexpl_09	16	(4)	13	(4)	29	(3)	Tak	1,071	Tak	1,277
pnbrexpl_12	14	(7)	13	(6)	32	(6)	Tak	8,522	Tak	1,051
pnbrexpl_15	13	(5)	12	(4)	32	(3)	Tak	1,025	Tak	0,802
PNwD	6	(6)	6	(6)	15	(15)	Tak	0,322	Tak	0,332
pn_campos_silva2	13	(10)	12	(9)	44	(7)	Tak	1,885	Tak	1,262
pn_campos_silva6	18	(18)	11	(11)	1	(1)	Nie	0,092	Nie	1,817
pn_campos_silva7	13	(9)	13	(9)	26	(11)	Tak	0,747	Tak	0,981
pn_desel_01	7	(3)	7	(3)	9	(2)	Tak	0,185	Tak	0,231
pn_desel_02	8	(8)	5	(5)	8	(8)	Tak	0,133	Tak	0,256
pn_desel_03	6	(6)	7	(7)	5	(5)	Tak	0,352	Tak	0,226
pn_fernandez3	21	(10)	20	(11)	20	(8)	Tak	0,554	Tak	3,554
pn_silva_01	9	(9)	9	(9)	6	(6)	Tak	0,147	Tak	0,517
pn_silva_02	6	(2)	4	(2)	4	(2)	Tak	0,084	Tak	0,163

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
pn_silva_03	14	(5)	10	(3)	32	(3)	Tak	0,965	Tak	0,846
pn_silva_04	5	(3)	6	(4)	6	(3)	Nieograniczona	0,031	Nieograniczona	0,183
pn_silva_05b	5	(4)	4	(3)	6	(4)	Nieograniczona	0,024	Nieograniczona	0,138
pn_silva_05c	5	(5)	3	(3)	3	(3)	Nieograniczona	0,024	Nieograniczona	0,111
pn_silva_05d	10	(10)	6	(6)	6	(6)	Tak	0,106	Tak	1,046
pn_silva_05e	4	(4)	4	(4)	4	(2)	Nieograniczona	0,022	Nieograniczona	0,109
pn_silva_05f	5	(5)	4	(4)	3	(3)	Tak	0,08	Tak	0,143
prio_ex	2	(2)	3	(2)	2	(2)	Tak	0,102	Tak	0,157
procesAM	29	(14)	25	(10)	74229	(0)	Nieograniczona	2977194,6	Nie	0,768
prod_cons	8	(4)	5	(2)	17	(0)	Nie	0,352	Nie	0,075
PUMA_loading	6	(2)	4	(1)	5	(0)	Nie	0,088	Nie	0,01
PUMA_unloading	5	(2)	3	(1)	4	(0)	Nie	0,127	Nie	0,008
PWM_extended	49	(2)	31	(2)	239	(2)	Tak	33,443	Tak	46,951
PWM_patterns	19	(2)	11	(2)	20	(2)	Tak	0,45	Tak	1,386
reactor_small	9	(4)	8	(4)	9	(3)	Tak	0,196	Tak	0,36

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
real-life_system_smarthome_simplified	12	(2)	7	(2)	10	(2)	Tak	0,28	Tak	0,442
reiseg1	13	(13)	9	(9)	17	(17)	Tak	0,511	Tak	0,813
rejers1	14	(2)	11	(2)	40	(2)	Tak	1,404	Tak	1,2
rejers2	11	(2)	10	(2)	12	(2)	Tak	0,255	Tak	0,504
return_books	4	(4)	6	(6)	4	(4)	Tak	0,082	Tak	0,185
RHINO_loading	6	(2)	4	(1)	5	(0)	Nie	0,111	Nie	0,01
RHINO_unloading	6	(2)	4	(1)	5	(0)	Nie	0,09	Nie	0,011
RSA_err	43	(32)	46	(35)	0	(0)	-	0	Nie	19,975
RSA_v1	49	(37)	58	(45)	21616	(15130)	Tak	539945,69	Tak	130627,9
RSA_v2	49	(37)	58	(45)	21616	(15130)	Tak	523429,78	Tak	125154,7
semaphore	3	(3)	4	(4)	3	(3)	Tak	0,075	Tak	0,131
semaphore2	9	(9)	6	(6)	5	(5)	Tak	0,104	Tak	0,303
SHA_decoder	200	(2)	170	(2)	0	(2)	-	0	Tak	4359,44
silva1	7	(3)	7	(3)	9	(2)	Tak	0,207	Tak	0,208
silva14	8	(4)	7	(3)	9	(2)	Tak	0,177	Tak	0,269

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
silva4v2	13	(10)	12	(10)	48	(19)	Tak	2,193	Tak	3,441
silva5	16	(16)	8	(8)	20	(20)	Tak	0,557	Tak	1,038
silva7	13	(9)	13	(9)	26	(11)	Tak	0,773	Tak	0,988
silva8	10	(7)	8	(6)	2	(1)	Nie	0,056	Nie	0,355
simple_production_system	9	(7)	6	(4)	12	(4)	Tak	0,236	Tak	1,467
sivaraman1	10	(2)	10	(2)	12	(2)	Tak	0,4	Tak	0,647
speedway	10	(2)	7	(2)	7	(2)	Tak	0,135	Tak	0,343
state_space16	16	(16)	16	(16)	256	(0)	Tak	167,934	Brak silnej spójności	8,77
state_space32	32	(32)	32	(32)	0	(0)	-	0	Brak silnej spójności	11,084
state_space48	48	(48)	48	(48)	0	(0)	-	0	Brak silnej spójności	39,466
sub-task_of_PLT_and_PMN_count	8	(6)	5	(3)	6	(0)	Nie	0,142	Nie	0,014
s_net_copy_milling_machine_subprocess	31	(5)	28	(6)	192	(4)	Tak	21,053	Tak	8,226
s_net_copy_milling_subprocess_quality	32	(5)	29	(6)	191	(5)	Nieograniczona	8,969	Nieograniczona	17,921
s_net_frame_manufact_quality_v1	17	(2)	16	(2)	21	(2)	Tak	0,652	Tak	3,174
s_net_frame_manufact_quality_v2_mod	19	(2)	18	(2)	30	(2)	Tak	1,596	Tak	1,939

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
tank_heating	4	(2)	4	(1)	4	(0)	Nie	0,075	Brak silnej spójności	0,039
TP5_I	8	(2)	6	(2)	8	(2)	Tak	0,178	Tak	0,222
traffic_lights	6	(5)	5	(4)	4	(3)	Tak	0,095	Tak	0,183
traffic_light_v1	8	(6)	3	(3)	3	(3)	Tak	0,08	Tak	0,233
traffic_light_v2	4	(2)	3	(2)	3	(2)	Tak	0,067	Tak	0,119
transfer	7	(6)	5	(4)	14	(0)	Nie	0,31	Nie	0,066
two_traffic_lights	7	(5)	6	(4)	5	(3)	Tak	0,094	Tak	0,213
vanDerAalst1	12	(7)	14	(8)	20	(8)	Tak	0,466	Tak	1,815
vanDerAalst3	14	(2)	13	(2)	22	(2)	Tak	0,472	Tak	0,86
vanDerAalst4	18	(2)	16	(2)	110	(2)	Tak	12,561	Tak	2,921
vanDerAalst5	23	(9)	23	(9)	63	(8)	Tak	5,356	Tak	5,488
vanDerAalst6	9	(6)	13	(9)	8	(5)	Tak	0,202	Tak	0,583
vanDerAalst7	14	(2)	13	(2)	42	(2)	Tak	1,451	Tak	0,873
vanDerAalst8	34	(19)	27	(14)	158	(21)	Tak	15,802	Tak	33,707
voorhoeve1	13	(8)	12	(7)	30	(6)	Tak	1,029	Tak	0,983

Nazwa sieci z bazy *Hippo*	Liczba miejsc (po redukcji)		Liczba tranzycji (po redukcji)		Liczba stanów (po redukcji)		Metoda bazowa		Proponowane rozwiązanie	
							Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]	Czy sieć jest żywa?	Czas wykonania weryfikacji [ms]
voorhoeve2	14	(8)	12	(7)	30	(6)	Tak	1,009	Tak	0,965
voorhoeve3	7	(2)	5	(2)	5	(2)	Tak	0,093	Tak	0,205
weijters1	12	(5)	14	(5)	20	(4)	Tak	0,548	Tak	0,794
well_built_alfa_net_copy_milling_machine_subprocess	30	(2)	25	(2)	187	(2)	Tak	18,616	Tak	6,064
xie1	24	(8)	26	(4)	128	(4)	Tak	14,284	Tak	5,476
zuberek1	30	(16)	22	(8)	136	(8)	Tak	13,266	Tak	7,022
zuberek3	29	(12)	21	(8)	208	(8)	Tak	26,537	Tak	20,163
zuberek4	30	(12)	21	(8)	208	(8)	Tak	28,872	Tak	5,944
zuberek5	41	(22)	31	(12)	176	(12)	Tak	20,686	Tak	16,979

Dodatek B. Wybrane kody źródłowe

A) Kod źródłowy metody umożliwiającej wyznaczenie składowych silnie spójnych (zawierających etykiety tranzycji).

```
std::vector<std::vector<int>> SCC::sccTransition() {
        for (int u = 0; u < graph.size(); u++) {
            if (!visited[u]) {
                dfsTransition(u);
            }
        }
        return componentsTransition;
    }

    void SCC::dfsTransition(int u) {
        lowLink[u] = time++;
        visited[u] = true;
        stack.push(u);
        bool isComponentRoot = true;

        for (int v : graph[u]) {
            if (!visited[v])
                dfsTransition(v);
            if (lowLink[u] > lowLink[v]) {
                lowLink[u] = lowLink[v];
                isComponentRoot = false;
            }
        }

        if (isComponentRoot) {
            std::vector component = std::vector<int>();
            while (true) {
                int x = stack.top();
                stack.pop();
                component.push_back(x);
                lowLink[x] = INT16_MAX;
                if (x == u) {
                    break;
                }
            }

                if(component.size()>1) {
                    auto transitionComponent = std::vector<int>();
                    for(int i : component) {
                        int ii = 0;
                        for(auto index: neighborhoodTransition[i]) {
                            if (std::find(component.begin(),
component.end(), graph[i][ii]) != component.end()) {

transitionComponent.push_back(neighborhoodTransition[i][ii]);
                            }
                            ii++;
                        }}

componentsTransition.push_back(transitionComponent);
            }
        }
    }
}
```

B) Kod źródłowy metod poszukujących sekwencji **S1**, **S2** oraz **S3**.

```
    bool PreliminaryVerification::findSequencesS1orS2() {
        for(int i=1; i<=petriNet.CountPlaces(); i++) {
            int l= 0;
            int nl= 0;
            for(int j=1; j<=petriNet.GetPlaceRow(i-1).size(); j++) {
                if(petriNet.GetPlaceRow(i-1)[j-1]==-1) {
                    l++;
                }
                if(petriNet.GetPlaceRow(i-1)[j-1]==1) {
                    nl++;
                }
            }
            if(l==1 && nl==0) {
                return true;
            }
        }
        return false;
    }

    bool PreliminaryVerification:: findSequencesS3() {
        for(int i=1; i<=petriNet.CountPlaces(); i++) {
            int  wejscia = 0;
            int wyjscia = 0;
            int index_t_we = -1;
            int index_t_wy = -1;
            for(int j=1; j<=petriNet.GetPlaceRow(i-1).size(); j++) {
                if(petriNet.GetPlaceRow(i-1)[j-1]==-1) {
                    wyjscia++;
                    index_t_wy = j-1;
                }
                if(petriNet.GetPlaceRow(i-1)[j-1]==1) {
                    wejscia++;
                    index_t_we = j-1;
                }
            }
            if(wejscia==1 && wyjscia==1) {
                for(int jj = 0; jj<
petriNet.GetTransitionRow(index_t_wy).size(); jj++) {
                   if(index_t_wy != -1 &&
petriNet.GetTransitionRow(index_t_wy)[jj] ==1) {
                      if(index_t_we != -1 &&
petriNet.GetTransitionRow(index_t_we)[jj] == -1) {
                         if(petriNet.GetMarkedPlaces()[jj] == 0 &&
petriNet.GetMarkedPlaces()[i-1] == 0) {
                             return true;
                         }
                      }
                   }
                }

                }

        }
        return false;
    }
```

C) Kod źródłowy metody umożliwiającej wyznaczanie grafu osiągalności stanów (wykorzystywany w proponowanym rozwiązaniu).

```
Void LivnessReachabilityGraph::generateReachabilityGraph() {
        states.AppendRow(pnets.GetMarkedPlaces());
        for(int i = 0; i<states.matrix().size();i++) {
            auto state = states.matrix()[i];
            auto transitionsIndexes  = findTransitionsFired(state);
            auto neighborhood = std::vector<int> ();
            auto transitions = std::vector<int> ();
            for(auto index : transitionsIndexes) {
                auto transitionState =
std::vector<int>(pnets.CountTransitions());
                transitionState[index] = 1;
                auto newState = generateNewState(state,
pnets.matrix().matrix()[index]);
                auto identicalRowIndexes = findIdenticalRow(states,
newState);
                if(identicalRowIndexes.empty()) {
                    states.AppendRow(newState);
                    transitionsStates.AppendRow(transitionState);
                    if (checkStateContainsNotBoundedPlace(newState,
i)) {
                        neighborhoodList = {};
                        neighborhoodListTransition = {};
                        return;
                    }
                    neighborhood.push_back(states.matrix().size()-
1);
                } else {
                    for(auto stateIndex :identicalRowIndexes) {
                        neighborhood.push_back(stateIndex);
                    }
                    transitionsStates.AppendRow(transitionState);
                }
                transitions.push_back(index);
            }
            neighborhoodList.push_back(neighborhood);
            neighborhoodListTransition.push_back(transitions);
        }
    }
```

Bibliografia

[1] W. van der Aalst, *Process Mining: Data Science in Action*, 2nd ed. Berlin Heidelberg: Springer-Verlag, 2016. doi: 10.1007/978-3-662-49851-4.

[2] W. M. P. van der Aalst, 'Decomposing Petri nets for process mining: A generic approach', *Distributed and Parallel Databases*, 2013, doi: 10.1007/s10619-013-7127-5.

[3] M. A. Alohali, F. N. Al-Wesabi, A. M. Hilal, S. Goel, D. Gupta, and A. Khanna, 'Artificial intelligence enabled intrusion detection systems for cognitive cyber-physical systems in industry 4.0 environment', *Cogn Neurodyn*, vol. 16, no. 5, pp. 1045–1057, Oct. 2022, doi: 10.1007/s11571-022-09780-8.

[4] P. S. Andrews and J. Timmis, 'Inspiration for the Next Generation of Artificial Immune Systems', in *Artificial Immune Systems*, C. Jacob, M. L. Pilat, P. J. Bentley, and J. I. Timmis, Eds., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 126–138. doi: 10.1007/11536444_10.

[5] A. Attoui, 'An environment based on rewriting logic for parallel systems formal specification and prototyping', *Journal of Systems Architecture*, vol. 44, no. 2, pp. 79–105, Nov. 1997, doi: 10.1016/S1383-7621(97)00003-9.

[6] M. Chen and R. Hofestädt, 'Quantitative petri net model of gene regulated metabolic networks in the cell', *Stud Health Technol Inform*, vol. 162, pp. 38–55, 2011.

[7] Y. Chen, Z. Li, and A. Al-Ahmari, 'Nonpure Petri Net Supervisors for Optimal Deadlock Control of Flexible Manufacturing Systems', *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 2, pp. 252–265, Mar. 2013, doi: 10.1109/TSMCA.2012.2202108.

[8] K. Chinzei, N. Hata, F. A. Jolesz, and R. Kikinis, 'MR Compatible Surgical Assist Robot: System Integration and Preliminary Feasibility Study', in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2000*, S. L. Delp, A. M. DiGoia, and B. Jaramaz, Eds., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2000, pp. 921–930. doi: 10.1007/978-3-540-40899-4_95.

[9] L. Dai and W. Guo, 'Concurrent Subsystem-Component Development Model (CSCDM) for Developing Adaptive E-Commerce Systems', in *Computational Science and Its Applications – ICCSA 2007*, O. Gervasi and M. L. Gavrilova, Eds., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 81–91. doi: 10.1007/978-3-540-74484-9_8.

[10] H. M. K. K. M. B. Herath and M. Mittal, 'Adoption of artificial intelligence in smart cities: A comprehensive review', *International Journal of Information Management Data Insights*, vol. 2, no. 1, p. 100076, Apr. 2022, doi: 10.1016/j.jjimei.2022.100076.

[11] H. Hu, M. Zhou, and Z. Li, 'Supervisor Design to Enforce Production Ratio and Absence of Deadlock in Automated Manufacturing Systems', *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 41, no. 2, pp. 201–212, Mar. 2011, doi: 10.1109/TSMCA.2010.2058101.

[12] St. Kirn, R. Unland, and U. Wanka, 'MAMBA: Automatic customization of computerized business processes', *Information Systems*, vol. 19, no. 8, pp. 661–682, Dec. 1994, doi: 10.1016/0306-4379(94)90035-3.

[13] R. W. Lewis, *Programming Industrial Control Systems Using IEC 1131-3*. IET Digital Library, 1998. doi: 10.1049/PBCE050E.

[14] A. Santone, V. Intilangelo, and D. Raucci, 'Application of Equivalence Checking in a Loan Origination Process in Banking Industry', in *2013 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Jun. 2013, pp. 292–297. doi: 10.1109/WETICE.2013.28.

[15] J. Xie, S. Liu, and X. Wang, 'Framework for a closed-loop cooperative human Cyber-Physical System for the mining industry driven by VR and AR: MHCPS', *Computers & Industrial Engineering*, vol. 168, p. 108050, Jun. 2022, doi: 10.1016/j.cie.2022.108050.

[16] C.-S. Shih, J.-J. Chou, N. Reijers, and T.-W. Kuo, 'Designing CPS/IoT applications for smart buildings and cities', *IET Cyber-Physical Systems: Theory Applications*, vol. 1, no. 1, pp. 3–12, 2016, doi: 10.1049/iet-cps.2016.0025.

[17] I. Grobelna, R. Wiśniewski, and M. Wojnakowski, 'Specification of Cyber-Physical Systems with the Application of Interpreted Nets', in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, Oct. 2019, pp. 5887–5891. doi: 10.1109/IECON.2019.8926908.

[18] M. Wojnakowski, M. Popławski, R. Wiśniewski, and G. Bazydło, 'Hippo-CPS: Verification of Boundedness, Safeness and Liveness of Petri Net-Based Cyber-Physical Systems', in *Technological Innovation for Digitalization and Virtualization*, L. M. Camarinha-Matos, Ed., in IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2022, pp. 74–82. doi: 10.1007/978-3-031-07520-9_7.

[19] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, Second edition. Cambridge, Massachuetts: The MIT Press, 2016.

[20] A. Barkalov, L. Titarenko, and K. Mielcarek, 'Improving characteristics of LUT-based Mealy FSMs', *AMCS*, vol. 30, no. 4, pp. 745–759, Dec. 2020, doi: 10.34768/amcs-2020-0055.

[21] A. G. Karatkevich and R. Wiśniewski, 'A polynomial-time algorithm to obtain state machine cover of live and safe Petri nets', *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 10, pp. 3592–3597, Oct. 2020, doi: 10.1109/TSMC.2019.2894778.

[22] A. R. Bakhtari, V. Kumar, M. M. Waris, C. Sanin, and E. Szczerbicki, 'Industry 4.0 Implementation Challenges in Manufacturing Industries: an Interpretive Structural Modelling Approach', *Procedia Computer Science*, vol. 176, pp. 2384–2393, Jan. 2020, doi: 10.1016/j.procs.2020.09.306.

[23] A. Ramirez-Trevino, I. Rivera-Rangel, and E. Lopez-Mellado, 'Observability of discrete event systems modeled by interpreted Petri nets', *IEEE Transactions on Robotics and Automation*, vol. 19, no. 4, pp. 557–565, Aug. 2003, doi: 10.1109/TRA.2003.814503.

[24] R. Wisniewski, I. Grobelna, and A. Karatkevich, 'Determinism in Cyber-Physical Systems Specified by Interpreted Petri Nets', *Sensors*, vol. 20, no. 19, Art. no. 19, Jan. 2020, doi: 10.3390/s20195565.

[25] I. Grobelna, 'Formal Verification of Control Modules in Cyber-Physical Systems', *Sensors*, vol. 20, no. 18, Art. no. 18, Jan. 2020, doi: 10.3390/s20185154.

[26] T. Tanida, T. Watanabe, and K. Onaga, 'A polynomial-time algorithm for finding a semi-generator of Petri net invariants', in *1991 IEEE International Symposium on Circuits and Systems (ISCAS)*, Jun. 1991, pp. 2838–2841 vol.5. doi: 10.1109/ISCAS.1991.176135.

[27] T. Murata, 'Petri nets: Properties, analysis and applications', *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989, doi: 10.1109/5.24143.

[28] W. M. P. van der Aalst, 'Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques', in *Business Process Management: Models, Techniques, and Empirical Studies*, W. van der Aalst, J. Desel, and A. Oberweis, Eds., in Lecture Notes in Computer Science. , Berlin, Heidelberg: Springer, 2000, pp. 161–183. doi: 10.1007/3-540-45594-9_11.

[29] P. Miczulski and M. Adamski, 'Analysis of safeness, liveness and persistence properties of petri nets by means of monotone logic functions', *IFAC Proceedings Volumes*, vol. 39, no. 17, pp. 137–142, 2006, doi: 10.3182/20060926-3-PL-4904.00023.

[30] L. A. Cortés, P. Eles, and Z. Peng, 'Modeling and formal verification of embedded systems based on a Petri net representation', *Journal of Systems Architecture*, vol. 49, no. 12, pp. 571–598, Dec. 2003, doi: 10.1016/S1383-7621(03)00096-1.

[31] M. Popławski, M. Wojnakowski, G. Bazydło, and R. Wiśniewski, 'Reachability Tree in Liveness Analysis of Petri Net-based Cyber-Physical Systems', in *AIP Conference Proceedings*, Heraklion, Greece, Jul. 2021.

[32] M. Wojnakowski, M. Popławski, R. Wiśniewski, and G. Bazydło, 'Hippo-CPS: Verification of Boundedness, Safeness and Liveness of Petri Net-Based Cyber-Physical Systems', in *Technological Innovation for Digitalization and Virtualization*, L. M. Camarinha-Matos, Ed., in IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2022, pp. 74–82. doi: 10.1007/978-3-031-07520-9_7.

[33] M. Wojnakowski, R. Wiśniewski, G. Bazydło, and M. Popławski, 'Analysis of safeness in a Petri net-based specification of the control part of cyber-physical systems', *AMCS*, vol. 31, no. 4, pp. 647–657, Dec. 2021, doi: 10.34768/amcs-2021-0045.

[34] S. Wang, M. Zhou, Z. Li, and C. Wang, 'A New Modified Reachability Tree Approach and Its Applications to Unbounded Petri Nets', *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 4, pp. 932–940, Jul. 2013, doi: 10.1109/TSMCA.2012.2226878.

[35] J. Martínez and M. Silva, 'A simple and fast algorithm to obtain all invariants of a generalised Petri net', in *Application and Theory of Petri Nets*, C. Girault and W. Reisig, Eds., in Informatik-Fachberichte. Berlin, Heidelberg: Springer, 1982, pp. 301–310. doi: 10.1007/978-3-642-68353-4_47.

[36] M. Wojnakowski and R. Wiśniewski, 'Verification of the Boundedness Property in a Petri Net-Based Specification of the Control Part of Cyber-Physical Systems', in *Technological Innovation for Applied AI Systems*, L. M. Camarinha-Matos, P. Ferreira, and G. Brito, Eds., in IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2021, pp. 83–91. doi: 10.1007/978-3-030-78288-7_8.

[37] R. Wiśniewski, G. Bazydło, L. Gomes, A. Costa, and M. Wojnakowski, 'Analysis and Design Automation of Cyber-Physical System with Hippo and IOPT-Tools', in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, Lisbon, Portugal: IEEE Press, Oct. 2019, pp. 5843–5848. doi: 10.1109/IECON.2019.8926692.

[38] 'Hippo / Petri nets'. Accessed: Jan. 07, 2022. [Online]. Available: http://hippo.issi.uz.zgora.pl/

[39] R. Wiśniewski, G. Bazydło, M. Wojnakowski, and M. Popławski, 'Hippo-CPS: A tool for verification and analysis of Petri net-based cyber-physical systems', in *Application and Theory of Petri Nets and Concurrency*, vol. 13929, L. Gomes and R. Lorenz, Eds., in Lecture Notes in Computer Science, vol. 13929. , Cham: Springer Nature Switzerland, 2023, pp. 191–204. doi: 10.1007/978-3-031-33620-1_10.

[40] T. M. Chen, J. C. Sanchez-Aarnoutse, and J. Buford, 'Petri Net Modeling of Cyber-Physical Attacks on Smart Grid', *IEEE Transactions on Smart Grid*, vol. 2, no. 4, pp. 741–749, Dec. 2011, doi: 10.1109/TSG.2011.2160000.

[41] X. He, Z. Dong, H. Yin, and Y. Fu, 'A Framework for Developing Cyber-Physical Systems', *Int. J. Soft. Eng. Knowl. Eng.*, vol. 27, no. 09n10, pp. 1361–1386, Nov. 2017, doi: 10.1142/S0218194017400010.

[42] E. A. Lee, 'Cyber Physical Systems: Design Challenges', in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, May 2008, pp. 363–369. doi: 10.1109/ISORC.2008.25.

[43] R. (Raj) Rajkumar, I. Lee, L. Sha, and J. Stankovic, 'Cyber-physical systems: the next computing revolution', in *Proceedings of the 47th Design Automation Conference on - DAC '10*, Anaheim, California: ACM Press, 2010, p. 731. doi: 10.1145/1837274.1837461.

[44] P. Gokhale, O. Bhat, and S. Bhat, 'Introduction to IOT', vol. 5, pp. 41–44, Jan. 2018, doi: 10.17148/IARJSET.2018.517.

[45] V. Lesch, M. Züfle, A. Bauer, L. Iffländer, C. Krupitzer, and S. Kounev, 'A literature review of IoT and CPS—What they are, and what they are not', *Journal of Systems and Software*, vol. 200, p. 111631, Jun. 2023, doi: 10.1016/j.jss.2023.111631.

[46] C.-S. Shih, J.-J. Chou, N. Reijers, and T.-W. Kuo, 'Designing CPS/IoT applications for smart buildings and cities', *IET Cyber-Physical Systems: Theory & Applications*, vol. 1, no. 1, pp. 3–12, 2016, doi: 10.1049/iet-cps.2016.0025.

[47] S. Li, L. D. Xu, and S. Zhao, 'The internet of things: a survey', *Inf Syst Front*, vol. 17, no. 2, pp. 243–259, Apr. 2015, doi: 10.1007/s10796-014-9492-7.

[48] D. Mrozek, A. Koczur, and B. Małysiak-Mrozek, 'Fall detection in older adults with mobile IoT devices and machine learning in the cloud and on the edge', *Information Sciences*, vol. 537, pp. 132–147, Oct. 2020, doi: 10.1016/j.ins.2020.05.070.

[49] J.-I. Latorre-Biel, J. Faulín, A. A. Juan, and E. Jiménez-Macías, 'Petri Net Model of a Smart Factory in the Frame of Industry 4.0', *IFAC-PapersOnLine*, vol. 51, no. 2, pp. 266–271, Jan. 2018, doi: 10.1016/j.ifacol.2018.03.046.

[50] B. Bagheri, S. Yang, H.-A. Kao, and J. Lee, 'Cyber-physical Systems Architecture for Self-Aware Machines in Industry 4.0 Environment', *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 1622–1627, Jan. 2015, doi: 10.1016/j.ifacol.2015.06.318.

[51] J. Lee, B. Bagheri, and H.-A. Kao, 'A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems', *Manufacturing Letters*, vol. 3, pp. 18–23, Jan. 2015, doi: 10.1016/j.mfglet.2014.12.001.

[52] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, 'The industrial internet of things (IIoT): An analysis framework', *Computers in Industry*, vol. 101, pp. 1–12, Oct. 2018, doi: 10.1016/j.compind.2018.04.015.

[53] S. Jeschke, C. Brecher, T. Meisen, D. Özdemir, and T. Eschert, 'Industrial Internet of Things and Cyber Manufacturing Systems', in *Industrial Internet of Things: Cybermanufacturing Systems*, S. Jeschke, C. Brecher, H. Song, and D. B. Rawat, Eds., in Springer Series in Wireless Technology. , Cham: Springer International Publishing, 2017, pp. 3–19. doi: 10.1007/978-3-319-42559-7_1.

[54] P. K. Verma *et al.*, 'Machine-to-Machine (M2M) communications: A survey', *Journal of Network and Computer Applications*, vol. 66, pp. 83–105, May 2016, doi: 10.1016/j.jnca.2016.02.016.

[55] J. Wan, M. Chen, F. Xia, L. Di, and K. Zhou, 'From machine-to-machine communications towards cyber-physical systems', *Computer Science and Information Systems*, vol. 10, no. 3, pp. 1105–1128, 2013.

[56] S. Leminen, M. Rajahonka, R. Wendelin, and M. Westerlund, 'Industrial internet of things business models in the machine-to-machine context', *Industrial Marketing Management*, vol. 84, pp. 298–311, Jan. 2020, doi: 10.1016/j.indmarman.2019.08.008.

[57] M. H. Miraz, M. Ali, P. S. Excell, and R. Picking, 'A review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT)', in *2015 Internet Technologies and Applications (ITA)*, Wrexham, United Kingdom: IEEE, Sep. 2015, pp. 219–224. doi: 10.1109/ITechA.2015.7317398.

[58] J.-R. Jiang, 'An improved cyber-physical systems architecture for Industry 4.0 smart factories', *Advances in Mechanical Engineering*, vol. 10, no. 6, p. 1687814018784192, Jun. 2018, doi: 10.1177/1687814018784192.

[59] V. Gourcuff, O. De Smet, and J.-M. Faure, 'Efficient representation for formal verification of PLC programs', in *2006 8th International Workshop on Discrete Event Systems*, Jul. 2006, pp. 182–187. doi: 10.1109/WODES.2006.1678428.

[60] E. R. Alphonsus and M. O. Abdullah, 'A review on the applications of programmable logic controllers (PLCs)', *Renewable and Sustainable Energy Reviews*, vol. 60, pp. 1185–1205, Jul. 2016, doi: 10.1016/j.rser.2016.01.025.

[61] H. Guo *et al.*, 'FPGA Implementation of VLC Communication Technology', in *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, Mar. 2017, pp. 586–590. doi: 10.1109/WAINA.2017.54.

[62] R. Wiśniewski, *Prototyping of Concurrent Control Systems Implemented in FPGA Devices*. in Advances in Industrial Control. Springer International Publishing, 2017. doi: 10.1007/978-3-319-45811-3.

[63] R. Wisniewski and I. Grobelna, 'Design of Multi-Context Reconfigurable Logic Controllers Implemented in FPGA Devices Oriented for Further Partial Reconfiguration', *J CIRCUIT SYST COMP*, vol. 27, no. 06, p. 1850086, Sep. 2017, doi: 10.1142/S021812661850086X.

[64] R. Wiśniewski, G. Bazydło, and P. Szcześniak, 'Low-Cost FPGA Hardware Implementation of Matrix Converter Switch Control', *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 7, pp. 1177–1181, Jul. 2019, doi: 10.1109/TCSII.2018.2875589.

[65] K. H. John and M. Tiegelkamp, *IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids*. Berlin, Heidelberg: Springer, 2010. doi: 10.1007/978-3-642-12015-2.

[66] J. L. Fernandez, R. Sanz, E. Paz, and C. Alonso, 'Using hierarchical binary Petri nets to build robust mobile robot applications: RoboGraph', in *2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA, USA: IEEE, May 2008, pp. 1372–1377. doi: 10.1109/ROBOT.2008.4543394.

[67] M. Szpyrka, *Sieci Petriego w modelowaniu i analizie systemów współbieżnych*. in Inżynieria Oprogramowania. Warszawa: Wydawnictwa Naukowo-Techniczne, 2008.

[68] C. A. Petri, 'Kommunikation mit Automaten', Mathematisches Institut der Universität Bonn, Bonn, 1962.

[69] A. Gogolińska, Ł. Mikulski, and M. Piątkowski, 'GPU Computations and Memory Access Model Based on Petri Nets', in *Transactions on Petri Nets and Other Models of Concurrency XIII*, vol. 11090, M. Koutny, L. M. Kristensen, and W. Penczek, Eds., in Lecture Notes in Computer Science, vol. 11090. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 136–157. doi: 10.1007/978-3-662-58381-4_7.

[70] J. Patalas-Maliszewska, M. Śliwa, and M. Topczak, 'Modelling the Demand for AM Technologies in Polish Manufacturing Enterprises Using Bayesian Networks', *Applied Sciences*, vol. 11, no. 2, Art. no. 2, Jan. 2021, doi: 10.3390/app11020601.

[71] L. Gomes and J. Ribeiro-Gomes, 'Analysing navigation paths in constrained graphs using Petri nets', in *2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE)*, Helsinki, Finland: IEEE, Jun. 2023, pp. 1–4. doi: 10.1109/ISIE51358.2023.10228055.

[72] M. Wojnakowski, R. Wisniewski, M. Poplawski, and G. Bazydlo, 'Analysis of Control Part of Cyber-Physical Systems Specified by Interpreted Petri Nets', in *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Prague, Czech Republic: IEEE, Oct. 2022, pp. 1090–1095. doi: 10.1109/SMC53654.2022.9945425.

[73] R. J. Wilson, *Introduction to graph theory*, 3rd ed., Repr. with minor corrections. Burnt Mill, Harlow, Essex, England : New York: Longman Scientific & Technical ; Wiley, 1985.

[74] C. Girault and R. Valk, *Petri Nets for Systems Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. doi: 10.1007/978-3-662-05324-9.

[75] R. David and H. Alla, 'Bases of Petri Nets', in *Discrete, Continuous, and Hybrid Petri Nets*, R. David and H. Alla, Eds., Berlin, Heidelberg: Springer, 2010, pp. 1–20. doi: 10.1007/978-3-642-10669-9_1.

[76] F. Dicesare, G. Harhalakis, J.-M. Proth, M. Silva-Suarez, and F. Vernadat, *Practice of Petri Nets in Manufacturing*. Springer Netherlands, 1993. doi: 10.1007/978-94-011-6955-4.

[77] Ł. Mikulski and I. Lanese, 'Reversing Unbounded Petri Nets', in *Application and Theory of Petri Nets and Concurrency*, S. Donatelli and S. Haar, Eds., Cham: Springer International Publishing, 2019, pp. 213–233. doi: 10.1007/978-3-030-21571-2_13.

[78] J. Hartmanis and R. E. Stearns, 'On the computational complexity of algorithms', *Trans. Amer. Math. Soc.*, vol. 117, pp. 285–306, 1965, doi: 10.1090/S0002-9947-1965-0170805-7.

[79] M. Wojnakowski, M. Popławski, R. Wiśniewski, and G. Bazydło, 'Safeness Analysis of Petri net-based Cyber-Physical Systems Based on the Linear Algebra and Parallel Reductions', in *AIP Conference Proceedings*, Heraklion, Greece, Jul. 2021.

[80] Marcin Wojnakowski, Remigiusz Wiśniewski, and Mateusz Popławski, 'Bounded and Place Invariant-Covered Petri Nets for Cyber-Physical Systems Specification', in *18th International Conference of Computational Methods in Sciences and Engineering Conference*, Heraklion, Crete, Greece, Oct. 2022.

[81] K. Jensen, 'How to find invariants for coloured Petri nets', in *Mathematical Foundations of Computer Science 1981*, J. Gruska and M. Chytil, Eds., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1981, pp. 327–338. doi: 10.1007/3-540-10856-4_100.

[82] K. Schmidt, 'On the Computation of Place Invariants for Algebraic Petri Nets', in *Structures in Concurrency Theory*, J. Desel, Ed., in Workshops in Computing. London: Springer, 1995, pp. 310–325. doi: 10.1007/978-1-4471-3078-9_21.

[83] K. Takano, S. Taoka, M. Yamauchi, and T. Watanabe, 'Experimental Evaluation of Two Algorithms for Computing Petri Net Invariants', *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E84-A, no. 11, pp. 2871–2880, Nov. 2001.

[84] R. Wiśniewski, J. Patalas-Maliszewska, M. Wojnakowski, and M. Topczak, 'Modeling and Analysis of a Petri Net-Based System Supporting Implementation of Additive Manufacturing Technologies', *IEEE Trans. Automat. Sci. Eng.*, pp. 1–11, 2023, doi: 10.1109/TASE.2023.3301304.

[85] R. Wiśniewski, M. Wojnakowski, and Z. Li, 'Design and Verification of Petri-Net-Based Cyber-Physical Systems Oriented toward Implementation in Field-Programmable Gate Arrays—A Case Study Example', *Energies*, vol. 16, no. 1, Art. no. 1, Jan. 2023, doi: 10.3390/en16010067.

[86] R. Wiśniewski, J. Patalas-Maliszewska, M. Wojnakowski, and M. Topczak, 'Interpreted Petri Nets in Modelling and Analysis of Physical Resilient Manufacturing Systems', in *2022 IEEE International Conference on Systems, Man, and*

Cybernetics (SMC), Oct. 2022, pp. 1096–1102. doi: 10.1109/SMC53654.2022.9945341.

[87] Marcin Wojnakowski, Remigiusz Wiśniewski, and Mateusz Popławski, 'A polynomial-time algorithm for detecting potentially unbounded places in a Petri net-based concurrent system', in *29th International European Conference on Parallel and Distributed Computing - Euro-Par*, Limassol, Cypr, August 28 – September 1.

[88] T. Hujsa and R. Devillers, 'On Deadlockability, Liveness and Reversibility in Subclasses of Weighted Petri Nets', *Fundamenta Informaticae*, vol. 161, no. 4, pp. 383–421, Jan. 2018, doi: 10.3233/FI-2018-1708.

[89] M. H. Abdul-Hussin and Z. A. Banaszak, 'On liveness and a class of generalized Petri nets', in *2017 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON)*, Bangkok, Thailand: IEEE, Aug. 2017, pp. 257–267. doi: 10.1109/IEMECON.2017.8079601.

[90] K. Barkaoui and M. Minoux, 'A polynomial-time graph algorithm to decide liveness of some basic classes of bounded Petri nets', in *Application and Theory of Petri Nets 1992*, vol. 616, K. Jensen, Ed., in Lecture Notes in Computer Science, vol. 616. , Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 62–75. doi: 10.1007/3-540-55676-1_4.

[91] G. Liu, Z. Li, and A. M. Al-Ahmari, 'Liveness Analysis of Petri Nets Using Siphons and Mathematical Programming', *IFAC Proceedings Volumes*, vol. 47, no. 2, pp. 383–387, 2014, doi: 10.3182/20140514-3-FR-4046.00078.

[92] A. M. Patel and A. Y. Joshi, 'Modeling and Analysis of a Manufacturing System with Deadlocks to Generate the Reachability Tree Using Petri Net System', *Procedia Engineering*, vol. 64, pp. 775–784, Jan. 2013, doi: 10.1016/j.proeng.2013.09.153.

[93] X. Ye, J. Zhou, and X. Song, 'On reachability graphs of Petri nets', *Computers & Electrical Engineering*, vol. 29, no. 2, pp. 263–272, Mar. 2003, doi: 10.1016/S0045-7906(01)00034-9.

[94] M. Hu, S. Yang, and Y. Chen, 'Partial Reachability Graph Analysis of Petri Nets for Flexible Manufacturing Systems', *IEEE Access*, vol. 8, pp. 227925–227935, 2020, doi: 10.1109/ACCESS.2020.3045980.

[95] Feng Chu and Xiao-Lan Xie, 'Deadlock analysis of Petri nets using siphons and mathematical programming', *IEEE Transactions on Robotics and Automation*, vol. 13, no. 6, pp. 793–804, Dec. 1997, doi: 10.1109/70.650158.

[96] M. Silva, J. M. Colom, J. Campos, and G. C, 'Linear Algebraic Techniques For The Analysis Of Petri Nets', in *In: Recent Advances in Mathematical Theory of Systems, Control, Networks, and Signal Processing II*, Mita Press, 1992, pp. 35–42.

[97] G. Liu, 'Structural Characteristics of Petri Nets', in *Petri Nets: Theoretical Models and Analysis Methods for Concurrent Systems*, G. Liu, Ed., Singapore: Springer Nature, 2022, pp. 35–68. doi: 10.1007/978-981-19-6309-4_2.

[98] G. Liu and K. Barkaoui, 'A survey of siphons in Petri nets', *Information Sciences*, vol. 363, pp. 198–220, Oct. 2016, doi: 10.1016/j.ins.2015.08.037.

[99] L. Jiao, T.-Y. Cheung, and W. Lu, 'Characterizing Liveness of Petri Nets in Terms of Siphons', in *Application and Theory of Petri Nets 2002*, vol. 2360, J. Esparza and C. Lakos, Eds., in Lecture Notes in Computer Science, vol. 2360. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 203–216. doi: 10.1007/3-540-48068-4_13.

[100] K. Barkaoui and M. Minoux, 'A polynomial-time graph algorithm to decide liveness of some basic classes of bounded Petri nets', in *Application and Theory of Petri Nets 1992*, K. Jensen, Ed., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1992, pp. 62–75. doi: 10.1007/3-540-55676-1_4.

[101] M. Popławski, R. Wiśniewski, G. Bazydło, and M. Maliński, 'Preliminary Verification of Liveness in a Control Part of Cyber-Physical Systems Modeled by

a Petri Net', in *Technological Innovation for Connected Cyber Physical Spaces*, vol. 678, L. M. Camarinha-Matos and F. Ferrada, Eds., in IFIP Advances in Information and Communication Technology, vol. 678. , Cham: Springer Nature Switzerland, 2023, pp. 205–215. doi: 10.1007/978-3-031-36007-7_15.

[102] Mateusz Popławski, Marcin Wojnakowski, Remigiusz Wiśniewski, and Grzegorz Bazydło, 'Initial Verification of Liveness Property in the Control Part of Cyber-Physical Systems modelled by Petri nets', in *18th International Conference of Computational Methods in Sciences and Engineering Conference*, Heraklion, Crete, Greece, Oct. 2022.

[103] R. Tarjan, 'Depth-first search and linear graph algorithms', in *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, East Lansing, MI, USA: IEEE, Oct. 1971, pp. 114–121. doi: 10.1109/SWAT.1971.10.

[104] R. H. Sloan and U. Buy, 'Reduction rules for time Petri nets', *Acta Informatica*, vol. 33, no. 5, pp. 687–706, Aug. 1996, doi: 10.1007/BF03036471.

[105] R. C. Martin and P. Gonera, *Czysty kod: poznaj najlepsze metody tworzenia doskonałego kodu*. Gliwice: Helion, 2023.

Spis rysunków

Spis tabel