

## Transforming nested structures of flowchart into hierarchical coloured Petri Nets

(*Work-in-Progress*)

Methawi Phokhai <sup>1,\*</sup>

Wiwat Vatanawood <sup>2,\*</sup>

---

\*Corresponding author

<sup>1</sup> Department of Computer Engineering, Faculty of Engineering Chulalongkorn University, Bangkok, Thailand, 6272071821@student.chula.ac.th

<sup>2</sup> Department of Computer Engineering, Faculty of Engineering Chulalongkorn University, Bangkok, Thailand, wiwat@chula.ac.th

### ABSTRACT

Flowchart is commonly used diagram to represent the processes in design phase of a software system. However, the flowchart of a complex software system inevitably contains the nested structures of branching and looping of the processes. The verification of these nested structure of the flowchart in advance is still difficult to conduct even using simulation techniques. In this paper, we intend to consider the complex flowchart with nested structures, so called nested-if and nested-loop, as our input design model. A set of mapping rules is proposed to transform the input complex flowchart with nested structures into the hierarchical coloured Petri nets to avoid the drawing of a single huge net of complicate model. The hierarchical coloured Petri nets also provides us to manage level of abstraction of the formal model and helps us concentrate on only an appropriate detail at a time. In our transforming approach, both data flow and control flow of the processes in flowchart are concerned as well so that all changing states of the observable variables in the flowchart would be represented and simulated in our resulting hierarchical coloured Petri nets. The CPN simulation tool is used to test and ensure the correctness of our resulting hierarchical coloured Petri nets.

*Keywords:* Flowchart, Nested structures, Formal verification, hierarchical coloured Petri nets.

### INTRODUCTION

In the software design phase, one of the common tools that are practically used to represent to process of the software system is flowchart. In order to assure the validity of the software system in the design phase, the formal verification method has been developed, such as model checking. However, one of the challenges in the complex systems is that it is impractical to search the exhaustively scenarios as to detect the unreachable path, or deadlock of these complex systems.

Numerous studies were proposed to transform the business models or UML models into the corresponding formal models and to enable the formal verification as to ensure the validity of their behaviors using LTL model checking. For example, (Deesukying, J., & Vatanawood, W., 2016) formalized the business rules in the business model into the chunk of colored Petri nets using CPN ML functions. (Maneerat, N., & Vatanawood, W., 2016) formally mapped the UML activity diagram in colored Petri nets. (Meghzili *et al.*, 2017) demonstrated the transforming of the UML state machine diagram into colored Petri nets using Isabelle/HOL. Several researches exploited the original Petri nets such as (Rocha, J. I. *et al.*, 2011) which proposed the dataflow model property verification using Petri nets.

For the flowchart, (Gulati, U., & Vatanawood, W., 2019) proposed a set of mapping rules to convert a basic flowchart into a colored Petri net including the types of colors of the tokens and their firing functions. In short, both data flow and basic control flow of the flowchart were concerned in order to visualize all changing states of all variables specified in the flowchart. In this paper, we propose the alternative transforming of the flowchart with the nested structures into hierarchical coloured Petri nets. The nested-if or nested-loop would be mapped into an appropriate hierarchical submodule within the outermost hierarchical coloured Petri nets.






### BACKGROUND

#### Flowchart

A flowchart (Chapin, N., 1970) represents a program flow using a set of symbols as a graphical model that shows the conceptual procedures or processes to the software owner. It is certainly used as a well-defined tool to demonstrate a step of a system such as sequence, decision, iteration, etc. The consecutive processes of the flowchart using square boxes connected with the arrow, called flowline, demonstrate a sequence of the workflow. Flowchart is used in the design and documentation phase of the software development which helps visualize the steps of work processes to get benefits of understanding and detect any problem in advance.

A simple flowchart consists of various symbols such as process, decision, and flowline (connected arrow) are shown in Table 1. Moreover, there are more symbols according to ANSI standard. In this paper, only basic symbols as shown in Table 1 are focused.

Table 1: The commonly used symbols in software documentation

Symbol	Name	Description
	Terminal	Represent the start and the end of the program
	Flowline	Represent the order of the program which indicates the sequences of the symbols
	Process	Represent the data transformation, data movement and logic operations
	Decision	Represent the decision of the program. This symbol makes 2 outputs by condition true or false
	Connector	Represent point of connection from other elements

**Coloured Petri Nets**

A coloured Petri net (Jensen, K., 1994) is developed using high level language to extend the ability to distinguish the data types of tokens called coloured set and explicitly embedded the user-defined functionality written in CPN ML source codes. For the complex system with both data flow and control flow, a coloured Petri net would be able to capture the dynamic properties of both data and control flows using coloured tokens and the Petri net’s control constructs. The CPN tool (Michael Westergaard and H.M.W. (Eric) Verbeek, 2015) has been developed to support the simulation and verification of coloured Petri nets. Moreover, in the CPN tool, the inscriptions which are the various labels, guards, conditions and user-defined function written in CPN ML programming language would be exploited to demonstrate the complex behaviors of the system. The various types of inscriptions are shown in Table 2. In our transforming approach, several inscriptions, such as guard inscription, code segment inscription, and initial marking inscription would be focused in order to represent the nested structures, nested-if and nested-loop, specified in a given flowchart.

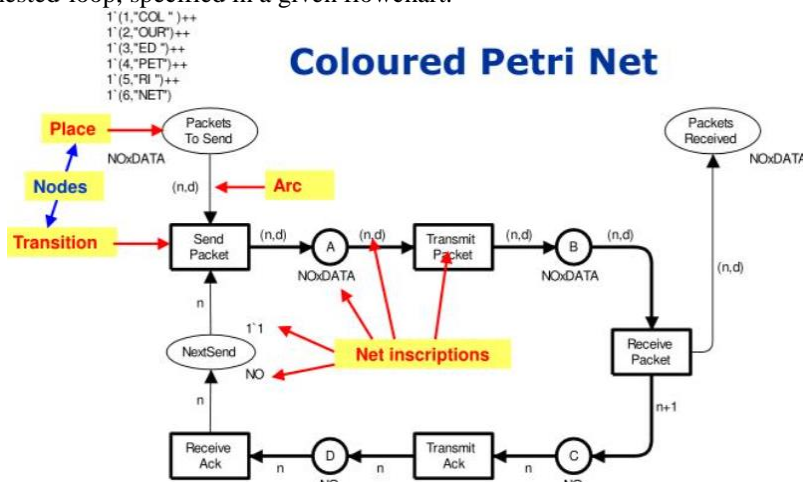


Figure 1: A sample of coloured Petri net components (Jensen, K., & Kristensen, L. M., 2009)

Table 2: Various Inscriptions of a coloured Petri net

Inscription	Name	Description
Transition	Transition Name Inscription	The label characters for state space tool
	Guard Inscription	The Boolean condition for expression
	Time Inscription	Using an @ symbol to delay expression
	Code Segment Inscription	The template to define the pattern for transition, consist of input (optional), output (optional), and code action (mandatory)
	Priority Inscription	The order of transition. (Should be non-negative integer expression)
Place	Color Set Inscription	The types of color set
	Initial Marking Inscription	The default value of place
	Place Name Inscription	The label name for place
Arc	Arc Inscription	The connection for place or transition, can be declare with single or multi element. It can declare Arc-Delay with time color set

### Hierarchical coloured Petri nets

For a complex software system, a formal model written in coloured Petri nets would commonly appear with the huge numbers of places and transitions and it would be difficult to understand. In CPN tool, hierarchical coloured Petri nets have been proposed to collapse a chunk of related places and transitions into submodule and replace this chunk with double lined rectangle to reduce the complexity of the original model. This hierarchy concept is a modelling technique that separates nets into a set of submodules which provide us to manage the complexity with level of abstraction.

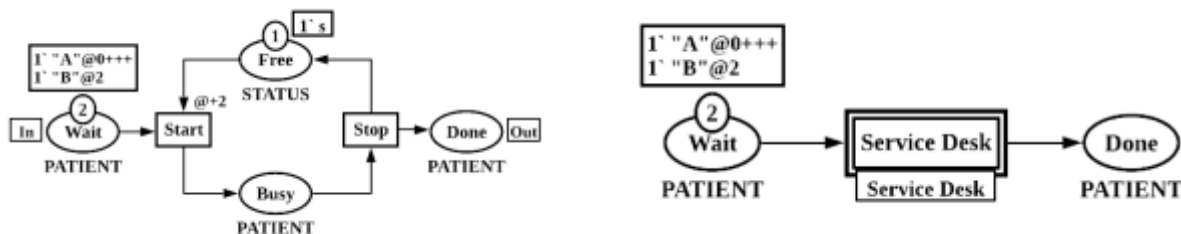


Figure 2: Service Desk module and Patient punch card module (Wil Van Der Aalst, M. P., & Stahl, C., 2011)

In Figure 2, the original coloured Petri net with non-hierarchy is shown on the left-hand side. Using the hierarchy concept, a chunk of places and transitions – transitions called Start and Stop, places called Free and Busy and their inscriptions, is collapsed into a separate submodule called Service Desk as shown on the right-hand side of the figure. Apparently, the new hierarchical coloured Petri net is more understandable and the submodule would be expanded as needed later.

The Service Desk tag below the Service Desk substitution transition is called substitution tag that shows the name of the submodule and, furthermore, inside it contains input place called input socket and output place called output socket. Then it needs to specify the substitution transition to complete the hierarchical coloured Petri nets for related interface of submodule. All can be done with port-socket relation which automatically assigns port place, input-output port, and substitution transition accordingly (Jensen, K., & Kristensen, L. M., 2009).

### TRANSFORMATION APPROACH

In this section, our transformation approach is described, and a set of mapping rules are presented in order to demonstrate the transforming of the symbols in flowchart into the corresponding chunks of places and their connecting transitions along with the mandatory inscriptions labelled in the resulting hierarchical coloured Petri nets.

Our approach begins with the importing of the original complex flowchart with nested structures – nested-if and nested-loop. However, we assume that the flowchart would be drawn in some specific steps. Firstly, the single initialization process symbol which defines all the necessary variables in the flowchart would be immediately drawn after the start symbol. Secondly, before each nested structure, there exists an index initialization process symbol. For example, the flowchart with two nested-loops is shown in Figure 3 and we can see that the system variable named “array” is defined in the initialization process symbol right after the start symbol. For the nested structures, an index initialization process symbol for this outermost nested-loop named “j=1” is drawn.

The original flowchart would be considered and the set of mapping rules are exploited to transform the symbols in the input flowchart into the corresponding hierarchical coloured Petri nets. The mapping rules concern with the compound colour set for array data structures and using code segment inscription and hierarchy tool for the nested structures. Finally, the CPN tool is used to verify the resulting hierarchical coloured Petri nets.

### Our transforming rules

The definitions of the flowchart and the hierarchical coloured Petri nets are described as follows.

#### Definition 1: Flowchart

A flowchart with the nested structure is formally defined as a 5-tuple  $FC = (S, PR, CN, D, E)$  where:

- S is the start symbol.
  - E is the end symbol.
  - PR is a set of process symbols and  $PR = SYSINIT \cup INDEXINIT \cup SIMPLE$ .
- where SYSINIT is a set of system initialization process symbol, INDEXINIT is a set of nested structure index initialization process symbol and SIMPLE is a set of the simple assignment process symbol.
- CN is a set of connector symbols.
  - D is a set of decision symbols.

Given a flowchart FC, each process symbol would be classified into SYSINIT, or INDEXINIT, or SIMPLE. At the moment, only one system initialization process symbol  $sinit \in SYSINIT$  is expected and before each nested structure found in the flowchart, there exists an index initialization process  $indinit \in INDEXINIT$ .

**Definition 2: hierarchical coloured Petri nets**

A hierarchical coloured Petri net is formally defined as a 14-tuple  $HCPN = (P, T, A, \Sigma, C, G, IM, V, L, CSEG, PT, HI, HO, TSUBM)$  where:

- $P$  is a finite set of places.
  - $T$  is a finite set of transitions such that  $P \cap T = \emptyset$ .
  - $A \subseteq P \times T \cup T \times P$  is a set of directed arcs.
  - $\Sigma$  is a finite set of non-empty colour sets.
  - $C: P \rightarrow \Sigma$  is a colour set function that assigns a colour set to each place.
  - $G: T \rightarrow EXPRV$  is a guard function that assigns a guard to each transition  $t$  such that  $Type[G(t)] = Bool$ .
  - $IM$  is the initial marking.
  - $V$  is a finite set of typed variables such that  $Type[v] \in \Sigma$  for all variables  $v \in V$ .
  - $L$  is a finite set of List variable such that  $L \in V$ .
  - $CSEG: T \rightarrow CODE$  is a code segment function that assigns a code segment inscription  $cs \in CODE$  to each transition  $t$  where  $CODE$  is the set of code segment inscription.
  - $PT: Pport \rightarrow \{IN, OUT, I/O\}$  is a port type function that assigns a port type to each port place  $pp \in Pport$  where  $Pport \subseteq P$ .
  - $HI$  is a finite set of input port place such that  $PT(pp) = IN$  where  $pp$  is a port place.
  - $HO$  is a finite set of output port place such that  $PT(pp) = OUT$  where  $pp$  is a port place.
  - $TSUBM \subseteq T$  is a set of substitution transitions where  $tsub \in TSUBM$  is a double line rectangle representing a submodule in the HCPN. A chunk of subnets of HCPN would be hidden within this submodule  $tsub$ .
- Given an input flowchart  $FC$  to be transformed into an output hierarchical coloured Petri nets HCPN, our mapping rules are shown as follow.

**Rule 1: Mapping start symbol**

A start symbol  $s$  in flowchart  $FC$  would be mapped to an initial place  $p \in P$  called “start” connected to a transition  $t \in T$  in HCPN.

**Rule 2: Mapping system initialization process symbol**

A system initialization process symbol  $sinit \in SYSINIT$  which is immediately appeared after the start symbol  $s$  in flowchart  $FC$  would be mapped to a place  $p \in P$  connected to a transition  $t \in T$ . All variables declared in the system initialization process symbol  $sinit$  would be collected and declared as the initial marking  $IM$  assigned in the initial place called “start”.

**Rule 3: Mapping nested structure**

If a nested structure index initialization process symbol  $nindex \in INDEX$  is detected in the give flowchart  $FC$ , the whole body of the nested structure in the flowchart  $FC$  would be mapped into a substitution transition  $tsub \in TSUBM$  as a submodule using double line rectangle symbol in HCPN. Both  $hi \in HI$  and  $ho \in HO$  would be created inside the submodule as the input port place and output port place respectively in order to locate the entry and exit points of the mentioned submodule/substitution transition  $tsub \in TSUBM$  using port-socket concept of the hierarchical coloured Petri nets. In this submodule  $tsub$ , a transition  $t \in T$  is created and connected to the input port place  $hi \in HI$  and a code segment inscription  $CSEG(t)=code \in CODE$  is assigned where  $code \in CODE$  describes how to handle the nested structure index.

For example, a nested structure initialization process symbol named “ $j=1$ ” is detected in the flowchart in Figure 3. According to this mapping rule, a substitution transition or submodule is created named “New Subpage” for the output hierarchical coloured Petri net as shown in Figure 4. The nested structure of flowchart is then mapped into the corresponding submodule of hierarchical coloured Petri net in Figure 5, where the input port place called “P3” and the output port place called “P4” are created and a transition named “TS1” is also created connected to the input port place named “P3”. The code segment handling the index  $j$  is assigned to the transition “TS1” as shown in Figure 5.

**Rule 4: Mapping simple process symbol**

A simple process symbol  $pr \in SIMPLE$  typically described the assignment statement of the system variables for the common calculation of the software system apart from the initialization process of the system variables and nested structure indexes. The simple process symbol  $pr$  would be mapped to a place  $p \in P$  connected to a transition  $t \in T$ . Any assignment statement in the process symbol  $pr$  would be converted into code segment inscriptions as well and assigned to the transaction  $t$  as  $CSEG(t) = code \in CODE$  where the code is written in the following format – input, output, action. For example, a simple process symbol named “ $i++$ ” of the flowchart in Figure 3 would be mapped to a place called “P5” and a transition called “T5” with code segment inscription.

**Rule 5: Mapping a connector symbol**

A connector symbol  $conn \in CN$  in flowchart  $FC$  would be mapped into place  $p \in P$  connected from the previous transition  $t \in T$  (if any). If there exist more than one previous transitions the place  $p$  would be connected from all of the previous transitions as well. For example, a connector following the process symbol named “ $i=1$ ” in the flowchart shown in Figure 3, would be mapped into a place called “CN1” in the hierarchical coloured Petri nets shown in Figure 4.

**Rule 6: Mapping a decision symbol**

A decision symbol  $d \in D$  in flowchart FC would be mapped into two transitions  $t1$ , and  $t2 \in T$ , where  $t1$  is expected to fire if the decision is True while the transition  $t2$  is expected to fire if the decision is False. The transition  $t1$  would be assigned with the guard function  $G(t1)$  according to the original condition written in the decision symbol  $d \in D$ . Whilst, the transition  $t2$  would be assigned with the guard function  $G(t2)$  according to the negation of the mentioned original condition. Then, each transition would be added with an arc as to be ready for the next rules.

For example, a decision symbol named “ $i < N$ ” in the flowchart as shown in Figure 3, would be mapped to the transition GT1, and GF1 where the transition GT1 is expected to fire if “ $i < N$ ” is True while the transition GF1 is expected to fire if “ $i < N$ ” is False.

**Rule 7: Mapping end symbol**

An end symbol  $e \in E$  in flowchart FC would be mapped into place  $p \in P$  called “End” connected to the previous transition  $t \in T$ . In our approach, only single end symbol is expected for the strictly well-formed style of the flowchart.

**CASE STUDY AND TOOL OVERVIEW****Bubble Sort**

Bubble Sort is the array sorting algorithm which is working by swapping the member of the array in the current position with the next position by ascending or descending order and repeat the process until reach  $N-1$  ( $N$  is the amount of the member of the array). The flowchart of the bubble sort is shown in Figure 3.

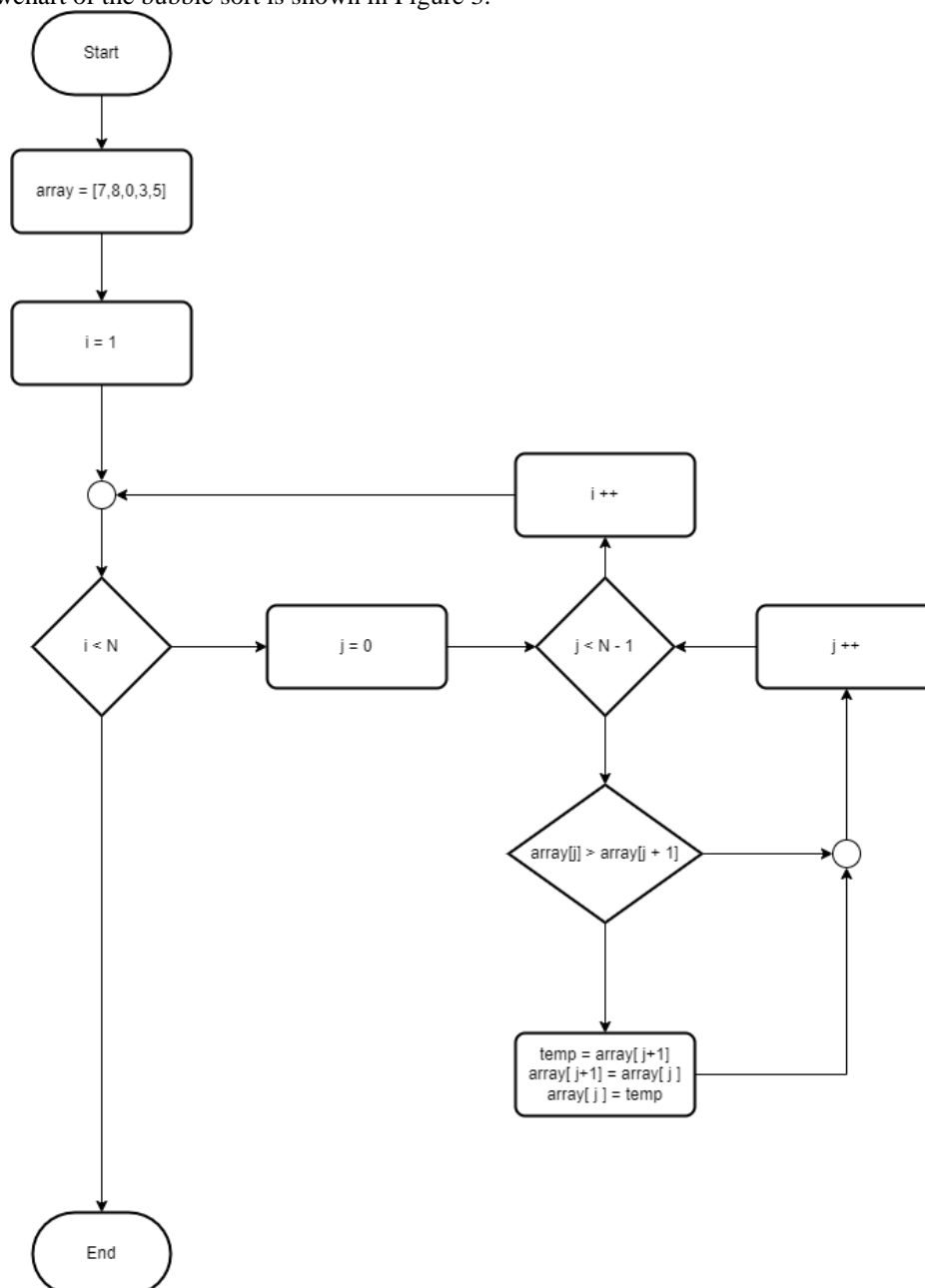


Figure 3: A flowchart created from bubble sort algorithm

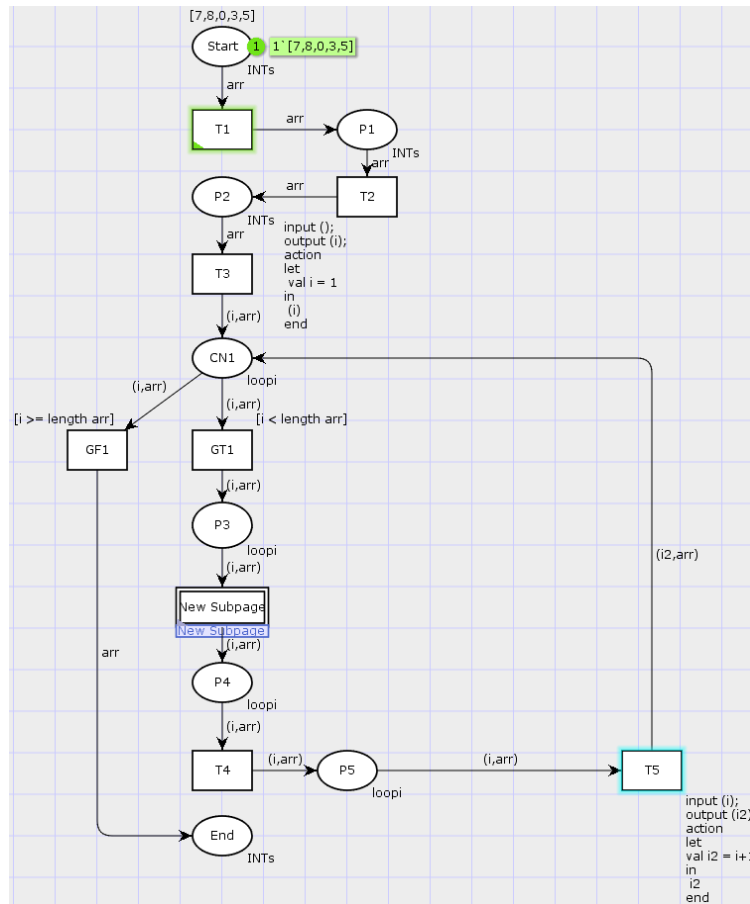


Figure 4: Loop i in bubble sort algorithm

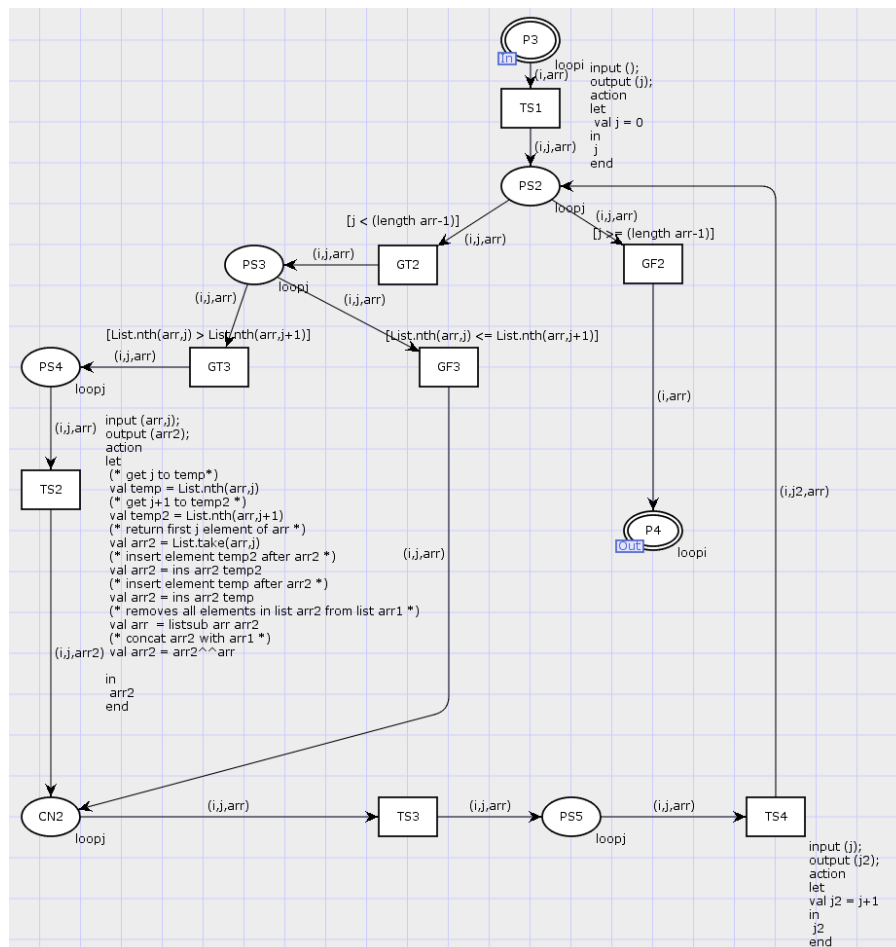


Figure 5: Loop j in bubble sort algorithm

Given a flowchart of bubble sort algorithm as shown in Figure 3, the corresponding hierarchical coloured Petri nets is transformed using our mentioned mapping rules. The highest level of net or main page is shown in Figure 4 and the lower level net of the submodule called “New Subpage” is shown in Figure 5. According to our mapping rules, the nested loop in the flowchart would be replaced with a submodule called “New Subpage”. The CPN tool is used to simulate and verify our hierarchical coloured Petri nets in Figure 4 and 5.

Firstly, the simulation shows us that the hierarchical coloured Petri nets are able to fire a token in comparable with the behaviors of the original flowchart of the bubble sort. Secondly, there are no unreachable places and transitions founded which means the mapping rules are sufficient. Thirdly, the value of the array to be sorted would be observable and correctly sorted after the simulation. The result of the bubble sort algorithm is shown in Figure 6 which appear as the array = [0,3,5,7,8].

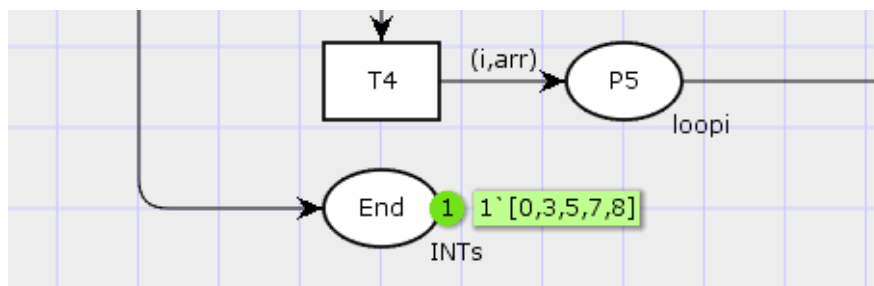


Figure 6: result of bubble sort algorithm

### CONCLUSION AND FUTURE WORK

In this paper, we propose an alternative transformation approach to convert a given complex flowchart with nested structures into the corresponding hierarchical coloured Petri nets using a set of mapping rules. In our transformation approach, the nested structure index is declared and handled locally using code segment inscriptions within the submodule in order to avoid the huge number of the global variable declaration. Both bubble sort and selection sort algorithms are used as our case studies to demonstrate the nested structure. The CPN tool is used to simulate and verify the correctness and the consistency of the system behaviors between the original flowchart and the output hierarchical coloured Petri nets. Moreover, the simulation shows both data flow and control flow of the original flowchart so that the changing states or values of all variables could be observed and traced step-by-step.

However, some limitations still exist for the future works. For example, the colour sets of the coloured Petri nets are focused only on boolean, integer, list, and character. The flowchart drawing style is assumed to have the system initialization process, nested structure index initialization process being classified.

### REFERENCES

- Chapin, N. (1970). Flowcharting with the ANSI standard: A tutorial. *ACM Computing Surveys (CSUR)*, 2(2), 119-146.
- Deesukying, J., & Vatanawood, W. (2016). Generating of business rules for coloured Petri Nets. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)* (pp. 1-6). IEEE.
- Draw.IO, (2018). [Online]. Available: <http://about.draw.io>.
- Gulati, U., & Vatanawood, W. (2019). Transforming flowchart into coloured Petri nets. In *Proceedings of the 2019 3rd International Conference on Software and e-Business* (pp. 75-80).
- Jensen, K. (1994). coloured Petri Nets Basic concepts. *Analysis Methods and Practical Use*, 3, 146-153.
- Jensen, K., & Kristensen, L. M. (2009). coloured Petri nets: modelling and validation of concurrent systems. Springer Science & Business Media.
- Jitmit, C., & Vatanawood, W. (2021). Simulating Artificial Neural Network Using Hierarchical coloured Petri Nets. In *2021 6th International Conference on Machine Learning Technologies* (pp. 127-131).
- Li, W. J., Yang, W., Li, S., & Liao, W. Z. (2012). The Algorithm of Color Petri Nets Transform into the Place/Transition Nets and Its Implementation. In *2012 11th International Symposium on Distributed Computing and Applications to Business, Engineering & Science* (pp. 461-465). IEEE.
- Maneerat, N., & Vatanawood, W. (2016). Translation UML activity diagram into coloured Petri net with inscription. In *2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)* (pp. 1-6). IEEE.
- Meghzili, S., Chaoui, A., Strecker, M., & Kerkouche, E. (2017). On the verification of UML state machine diagrams to colored petri nets transformation using Isabelle/HOL. In *2017 IEEE International Conference on Information Reuse and Integration (IRI)* (pp. 419-426). IEEE.
- Michael Westergaard and H.M.W. (Eric) Verbeek (2015). Retrieved from <https://cpntools.org/>
- Rocha, J. I., Gomes, L., & Dias, O. P. (2011). Dataflow model property verification using Petri net translation techniques. In *2011 9th IEEE International Conference on Industrial Informatics* (pp. 783-788). IEEE.
- Thampibal, L., & Vatanawood, W. (2019). Formalizing Railway Network Using Hierarchical Timed coloured Petri Nets. In *Proceedings of the 2019 7th International Conference on Information Technology: IoT and Smart City* (pp. 338-343).
- Wil Van Der Aalst, M. P., & Stahl, C. (2011). Modeling business processes: a petri net-oriented approach. MIT press.