When Graffiti Brings Order*

Alban Linard and Didier Buchs

Université de Genève, Centre Universitaire d'Informatique
7 route de Drize, 1227 Carouge, Suisse

Abstract. Most researchers use Petri nets as a formal notation with mathematically defined semantics. Their graphical part is usually only seen as a notation, that does not carry semantics.
Contrary to this tradition, we show in this article that, when created by a human, there is inherent semantics in the positions of places, transitions and arcs. We propose to use the full definition of Petri nets: whereas they have been deteriorated to their mathematically defined part only, their graphical information should be considered in their definition.

1 Introduction

Graphical information of Petri nets usually does not appear in their formal definitions. For instance, Figure 1 presents the traditional graphical Petri net representation of two dining philosophers, a textual, and a mathematical representation of the same Petri net. In research, we almost always consider them as equivalent. The graphical representation is used by the modeler, or for figures in articles, the textual representation is an example of input format for a tool, and the mathematical representation is used for scientific publications.

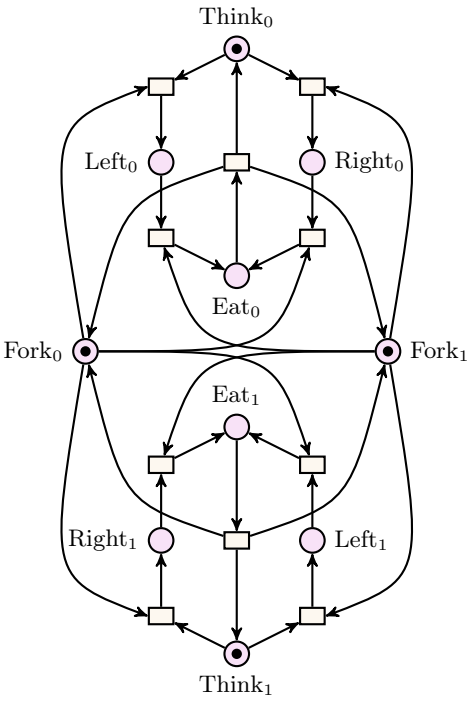
Are all these representations really equivalent? When we ask researchers in Petri nets, they often believe it. But their equivalence is only an assumption, it might thus be false. Why do people strive to maintain layouts in the model transformation field, for instance in [1]? Our doubts for Petri nets are exposed in Dialogue 1, through a fictional dialogue between two elements of a Petri net:

MASTER TRANSITION: Why does graphical information of Petri nets disappear in their formal definitions?
PLACEHOPPER: Because graphical information has no semantics.
MASTER TRANSITION: If there is no semantics in graphical information, why are Petri nets represented graphically?
PLACEHOPPER: Because it helps the modelers to write and understand the models.
MASTER TRANSITION: But how does graphical information help the modelers if it has no semantics?

Dialogue 1: Question of PLACEHOPPER

Throughout this article, we propose to investigate the problem raised by MASTER TRANSITION, by extracting semantics from the graphical part of Petri

*Thanks to Matteo Risoldi for proposing this title.




```
Places
  Think0, Think1 = 1
  Fork0 , Fork1  = 1
  Left0, Right0, Eat0 = 0
  Left1, Right1, Eat1 = 0
Transitions
  Think0 + Fork0  → Left0
  Think0 + Fork1  → Right0
  Left0  + Fork1  → Eat0
  Right0 + Fork0  → Eat0
  Eat0  → Fork0 + Fork1 + Think0
  Think1 + Fork1  → Left1
  Think1 + Fork0  → Right1
  Left1  + Fork0  → Eat1
  Right1 + Fork1  → Eat1
  Eat1  → Fork0 + Fork1 + Think1
```

$$PN = \langle P, TA \rangle$$

$$P = \left\{ \begin{array}{l} Think_0, Left_0, Right_0, Eat_0, Fork_0 \\ Think_1, Left_1, Right_1, Eat_1, Fork_1 \end{array} \right\}$$

$$T = \left\{ \begin{array}{l} r_0, s_0, t_0, u_0, v_0 \\ r_1, s_1, t_1, u_1, v_1 \end{array} \right\}$$

$Pre =$

	$Think_0$	$Left_0$	$Right_0$	Eat_0	$Fork_0$	$Think_1$	$Left_1$	$Right_1$	Eat_1	$Fork_1$
r_0	1	0	0	0	1	0	0	0	0	0
s_0	1	0	0	0	0	0	0	0	0	1
t_0	0	1	0	0	0	0	0	0	0	1
u_0	0	0	1	0	1	0	0	0	0	0
v_0	0	0	0	1	0	0	0	0	0	0
r_1	0	0	0	0	0	1	0	0	0	1
s_1	0	0	0	0	1	1	0	0	0	0
t_1	0	0	0	0	1	0	1	0	0	0
u_1	0	0	0	0	0	0	0	1	0	1
v_1	0	0	0	0	0	0	0	0	1	0

$Post =$

	$Think_0$	$Left_0$	$Right_0$	Eat_0	$Fork_0$	$Think_1$	$Left_1$	$Right_1$	Eat_1	$Fork_1$
r_0	0	1	0	0	0	0	0	0	0	0
s_0	0	0	1	0	0	0	0	0	0	0
t_0	0	0	0	1	0	0	0	0	0	0
u_0	0	0	0	1	0	0	0	0	0	0
v_0	1	0	0	0	1	0	0	0	0	1
r_1	0	0	0	0	0	0	1	0	0	0
s_1	0	0	0	0	0	0	0	1	0	0
t_1	0	0	0	0	0	0	0	0	1	0
u_1	0	0	0	0	0	0	0	0	1	0
v_1	0	0	0	0	1	1	0	0	0	1

Fig. 1: Graphical, textual and mathematical representations of a Petri net

nets. This semantics is unclear, as it depends on the modeler and the model. Thus, we check that the extracted semantics can be useful to improve model checking performance of the Petri nets.

Section 2 presents the methodology of this work: how semantics extracted from the graphical information is used to improve model checking performance. Then we propose to use alignments of places and transitions in Section 3. As this only information is not always sufficient to improve model checking performance, we also propose to use graphical distances in Section 3. Then we propose in Section 4 to use surfaces delimited by arcs, for models with no alignments. The approach is generalized to colored Petri nets in Section 5. As we cannot give semantics to graphical information of all models, two counter-examples are provided in Section 6. Section 7 discusses about when and how graphical information can be used, before conclusion in Section 8.

2 How do we assess the quality of semantics extracted from graphical information?

The Software Modeling and Verification Group has developed the **Al**gebraic **P**etr**i** **N**et **A**nalyzer (AlPiNA) [2], a model checker for Algebraic Petri nets. For efficient state space computation, this tool is based on Decision Diagrams (DDs) [3,4]. In this approach, a Decision Diagram, which is a particular kind of Directed Acyclic Graph, represents the state space. A node in the graph, called "variable", represents the marking of each color for each place. When using DDs, the efficiency highly depends on the variable order in the graph [5].

By default, AlPiNA has rather good computation times [6], but its efficiency can be improved by orders of magnitude when the user provides "clustering" information, to group related variables. It is given in a textual notation. For instance, Listing 1 shows a clustering for the `Philosophers` model[1]. Variables for the black token ● in places `Think0 .. Eat0` are put in the same cluster (group of variables) `c0`. The same applies for ● in `Think1 .. Eat1`, put in cluster `c1`.

```
Clusters c0, c1;
Rules
  cluster of ●
         in Think0, Left0, Right0, Fork0, Eat0
         is c0;
  cluster of ●
         in Think1, Left1, Right1, Fork1, Eat1
         is c1;
  c0 < c1;
```

Listing 1: Example of clustering

Several articles already give heuristics to infer variable orders from Petri nets [7]. Clustering is less precise, as it only defines groups of variables, and

[1] This is not the exact syntax used in AlPiNA, but a very near one.

groups order. It does not define the order of variables within each group. However, AlPiNA also allows to define an order over places.

In this article, we propose to use the graphical information of Petri nets to define the clustering, together with ordering of the variables in the clusters. We try to group together tokens and places that belong to the same process.

To assess the inferred clustering, we check if it improves the model checker performances. Instead of using AlPiNA, we use `PNXDD` [8]. It is another Decision Diagram-based model checker that provides several clustering and ordering algorithms, contrary to AlPiNA. By using it, we can easily compare our approach with these algorithms. Because they are simpler, we begin our experiments with Place/Transition Petri nets and then use a Symmetric Petri net.

All the examples in this article are taken from the Model Checking Contest of the SUMo 2011 workshop. Choosing models that were not created by our group allows us to avoid a possible bias that may be introduced if we had too much knowledge about the models.

For each model, we compare the efficiency of our graphical clustering and ordering with fully random ones (500 random clusterings and orderings for each clustering proposed in this article). This benchmark shows the experimental probability of obtaining clustering and ordering with the same efficiency. Then we compare our graphical clustering and ordering with all the algorithms implemented in `PNXDD`. It shows how we perform against several existing algorithms.

The results are not intended to show that our approach is more efficient than other ordering algorithms, for instance those used by [9]. We provide them primarily to assess how much semantics is put in the graphical information of Petri nets, and also to assess if the semantics we extract makes sense. Therefor, we propose throughout this article informal ways to extract semantics from the graphical representation of Petri nets, instead of well defined algorithms.

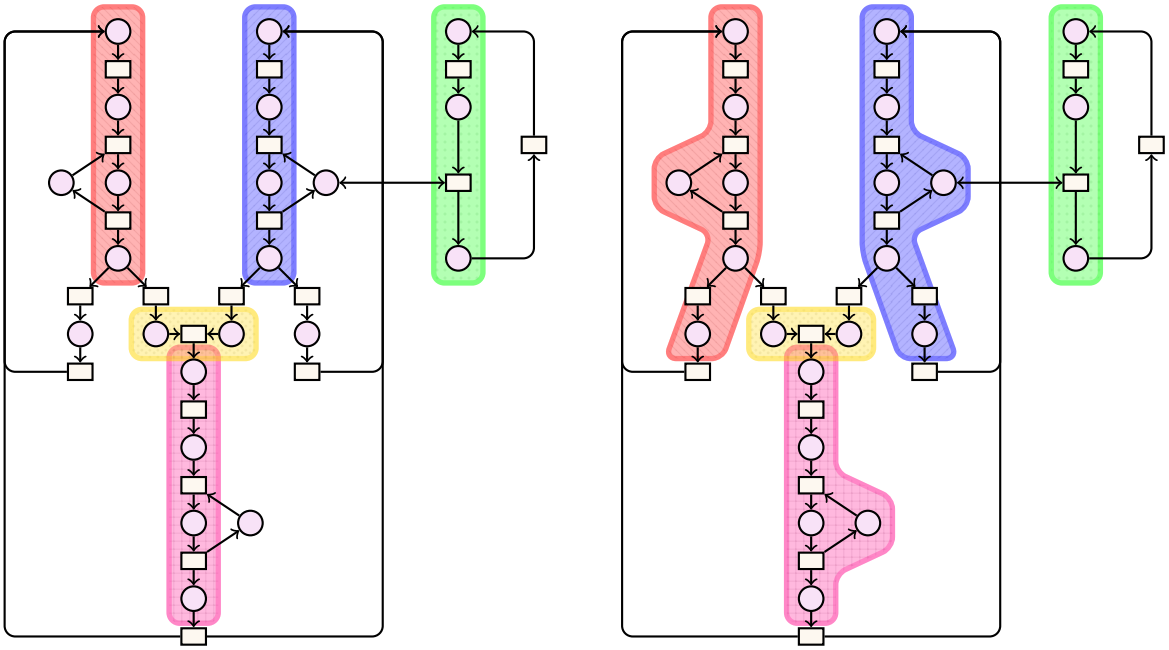
We provide a VirtualBox image of the tools and scripts used in the benchmarks at `http://smv.unige.ch/members/dr-alban-linard/sumo2011-when-graffiti-brings-order-image` (use `login/pass: sumo2011` for the Linux session).

3 Using alignment of places and transitions

We start the exploration of usual graphical semantics in Petri nets with the `Flexible Manufacturing System` (`FMS`) model. It is a Place/Transition Petri net represented in Figure 2. Initial marking is not shown as it is irrelevant in our approach. Note that this model is parameterized by its initial marking. The full model is presented in the SUMo 2011 workshop.

3.1 Alignment only

Even with absolutely no knowledge of the modeled system, it is obvious that the PN has vertically aligned places and transitions, linked by arcs in the alignment. Alignment, either vertical or horizontal, is one usual way for the modelers to show graphically the processes. Figure 2a shows the four vertical



(a) Aligned places and transitions (b) With places added to nearest group

Fig. 2: `Flexible Manufacturing System`

alignments of places and transitions, and one horizontal alignment. Thus, maximal non-overlapping chains of consecutive and aligned places and transitions define groups. All the places of each group can be put in a same cluster. From this example, we can state Assumption 1:

Assumption 1 *Aligned places and transitions may correspond to a process.*

3.2 Alignment with graphical distance

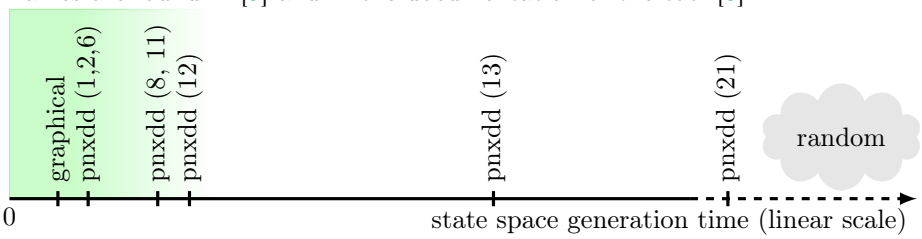
After applying Assumption 1, some places do not belong to any group. Usually, the modelers use them either to represent shared data or for local data. In Figure 2a, the horizontal alignment shows a shared data. The remaining marked places are each near the middle of a cluster, and thus are likely to represent local data. On the contrary, the remaining unmarked places are found at the extremity of the clusters, and thus they can either represent the processes or local data. So, after defining the groups, we extend them using Assumption 2:

Assumption 2 *Local data of a process is more likely to be near its process.*

Figure 2b shows the result of application of Assumption 2 to the groups of Figure 2a. Each place is added to the nearest group it is related to, where nearest means graphical distance. For instance, the unmarked places are added to the vertical groups instead of the horizontal one, because they are linked to transitions aligned with the vertical clusters.

We also define ordering of the places inside the clusters and ordering of the clusters. Inside each cluster, the places are ordered from top to bottom or from left to right. The local data places are put randomly before or after the nearest process place. To order the clusters, we also chose graphical distance. In the `FMS` model,  and are next to each other.  is far from all other clusters, but its nearest neighbor is  . As is between  and  , we get the following order for the clusters:

The results of the benchmarks for the `FMS 50`, *i.e.*, with initial marking parameter set to 50. All random clusters and orders were too slow, and thus could not finish computation. Graphical clustering and ordering gives better results than all algorithms implemented in `PNXDD`. We give only the number of the algorithm, instead of its name, for a concise diagram. The corresponding algorithm names are found in [9] and in the documentation of the tool [8].



4 Using surfaces

Unlike the `fms` model, the `kanban` model shown in Figure 3 contains almost no alignments. The modeler has used different graphical semantics for this model.

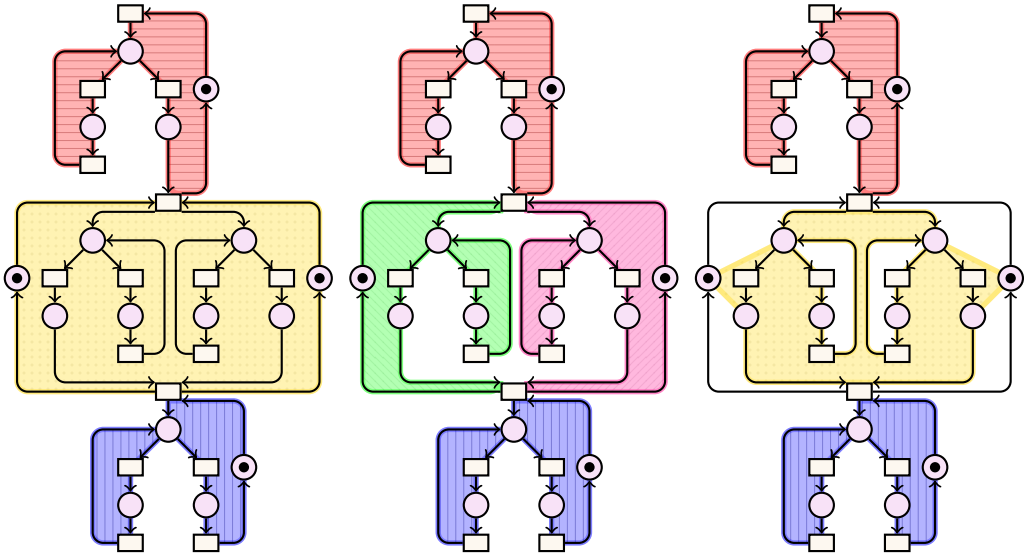
For the human eye, the `kanban` model is composed of four components, each one with four places. All components have rather similar shapes. They differ by some arcs and also by a symmetry. Detecting these small differences is not a trivial task.

The components are visible because the arcs draw the shapes of the components. For instance, the modeler used arcs with two inflexion points where he could have drawn a direct arc. From this example, we can deduce Assumption 3:

Assumption 3 *Arcs may draw shapes, the surface of which may contain related places.*

We investigate different ways to use the shapes. First, Figure 3a identifies three different components in the model. Each one is found by searching maximal surfaces in the graphical Petri net. When two surfaces are linked by a place, they are merged, as one place can only belong to one cluster.

The maximal surfaces approach can lead to huge clusters. The worst case is when the whole Petri net is enclosed by arcs, and thus all its places are in the same cluster. We thus propose a second way to search surfaces. Instead of identifying maximal surfaces, we can use minimal surfaces. They are defined

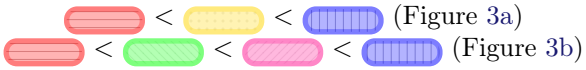


(a) Maximal surfaces (b) Minimal surfaces (c) Minimal connected surfaces with nearest remaining places

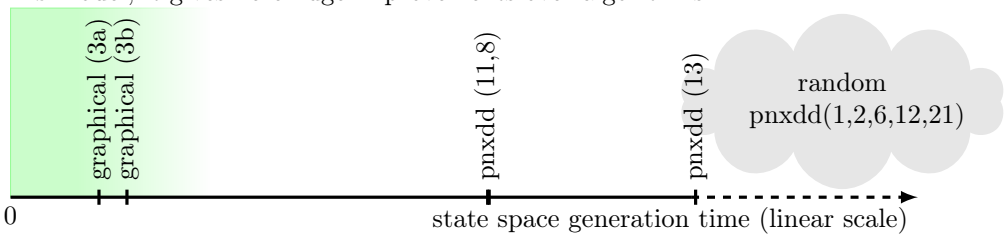
Fig. 3: `Kanban System`

as the surfaces that do not enclose another surface. Figures 3b and 3c show the `Kanban` model with minimal surfaces. In Figure 3b, the choice of minimal surfaces to consider leads to four clusters, whereas in Figure 3c another choice leads to three clusters. As the surfaces that touch the same place must be merged, we have to choose either Figure 3b or Figure 3c. Otherwise, the three surfaces in the middle block are merged, and the clustering is equivalent to the one of Figure 3a.

We compare Figure 3a and Figure 3b in the benchmarks. Places are ordered inside the clusters by following the boundaries of minimal surfaces. Clusters are ordered by graphical proximity, giving the clusters below:



The results of the benchmarks for the `Kanban` model are given below. All random clusters and orders were again too slow, and thus could not finish computation. The same applies to some algorithms of `PNXDD` (1,2,6,12,21). Whereas graphical clustering shows a small improvement over existing algorithms for the `FMS` model, it gives here huge improvements over algorithms in `PNXDD`.



5 Extension to colored Petri nets

The `Shared Memory` model of the SUMo 2011 workshop is given in Figure 4. It is a Symmetric Petri net, which is a particular kind of colored Petri net. Of course, our approach can be extended to the unfolded Place/Transition Petri net equivalent to a colored Petri net, when this equivalence exists. But the unfolding must then generate positions (possibly in three or more dimensions) to avoid superposition of unfolded places and transitions.

Without unfolding, we can still in some cases use graphical information to define clusters. Clusters for colored Petri nets are groups of unfolded places, *i.e.,* places together with colors. Two policies can coexist:

- Grouping in the same cluster all colors for one place,
- Distributing in its own cluster each color for one place.

The first policy works usually well for places representing shared data, whereas the second one leads usually to good results for places representing processes or data local to a process. Assumptions 4 and 5 describe these two policies.

Assumption 4 *One place related to other places using different variables may represent a shared data.*

Assumption 5 *Several places related to each other using the same variable, or equal ones, may represent a process.*

In the `Shared Memory` model, both Assumption 1 and Assumption 3 can be applied, as we find both aligned places and transitions and surfaces.

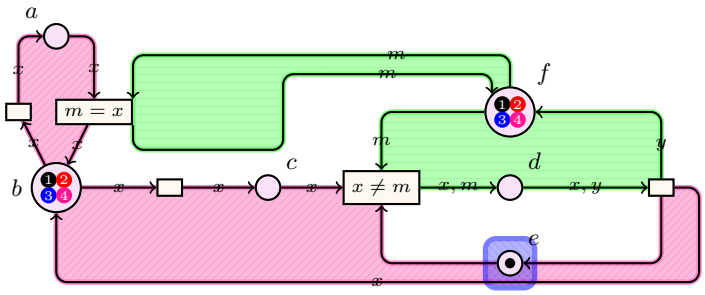
5.1 Using surfaces

To consider surfaces, we use Assumption 3. Its clustering is given in Figure 4a. We divided the model in four minimal surfaces. As there are places common to several surfaces, we obtain two groups. All places (a, b, c, e) in the group have the same domain, except one (e) that is not colored (shown using). We create one cluster for each color ❶, ❷, ❸, or ❹, as shown in Listing 2. Place e can only belong to one cluster, as its domain is the black token ●. Thus, we also create its own separate cluster.

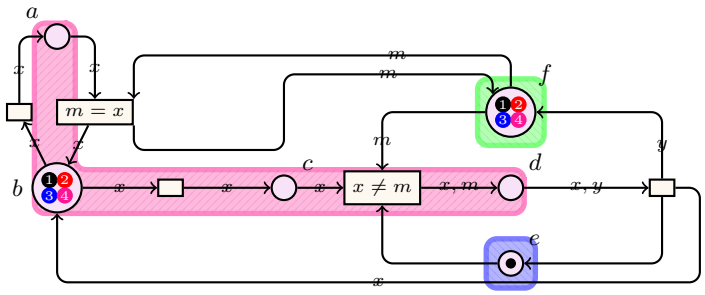
```
Clusters c, c1, c2, c3, c4;
Rules
  cluster of ● in e is c;
  cluster of ❶ in a,b,c is c1;
  cluster of ❷ in a,b,c is c2;
  cluster of ❸ in a,b,c is c3;
  cluster of ❹ in a,b,c is c4;
```

Listing 2: Clustering using surfaces for places a, b, c, e

The group contains two places: d and f. Place f has domain $\{❶, ❷, ❸, ❹\}$, whereas the domain of the place d is a couple of colors. For each color, we create one cluster. Listing 3 completes Listing 2 with the new clusters.



(a) Minimal surfaces and Identity, with nearest Identity



(b) Alignment and Identity, with nearest Identity

Fig. 4: `Shared Memory`

Each token in place f is put in the cluster of its color. In place d, each token is a couple of colors. We have to choose which component of the couple predominates. Here, there is no obvious choice, so we choose the first element. Thus all tokens in place d are put in the cluster of the color of the first component.

```
Clusters d1, d2, d3, d4;
Rules
  cluster of ❶ in f is d1;
  cluster of ❷ in f is d2;
  cluster of ❸ in f is d3;
  cluster of ❹ in f is d4;
  cluster of {❶} × {❶,❷,❸,❹} in d is d1;
  cluster of {❷} × {❶,❷,❸,❹} in d is d2;
  cluster of {❸} × {❶,❷,❸,❹} in d is d3;
  cluster of {❹} × {❶,❷,❸,❹} in d is d4;
  c < c1 < d1 < c2 < d2 < c3 < d3 < c4 < d4;
```

Listing 3: Clustering using surfaces for places d, f

5.2 Using alignments

Using Assumption 1 gives a totally different result. It is shown in Figure 4b. Listing 4 shows the clustering obtained using Assumption 1.

We first identify aligned places and transitions: places b, c, d belong to one group. Places b and c have the same colored domain, whereas the tokens of place d are couples of colors.

We then try to add to the group what seems local data or part of the processes. All places a, f, e are near the group. Place a is linked to the group using only variable x. This clearly shows that an identity is passed along the arcs. So, this place is added to the group.

Place e contains black tokens. Because of this domain, it cannot be added in the clusters of colors, so we create its own cluster, as in Listing 2.

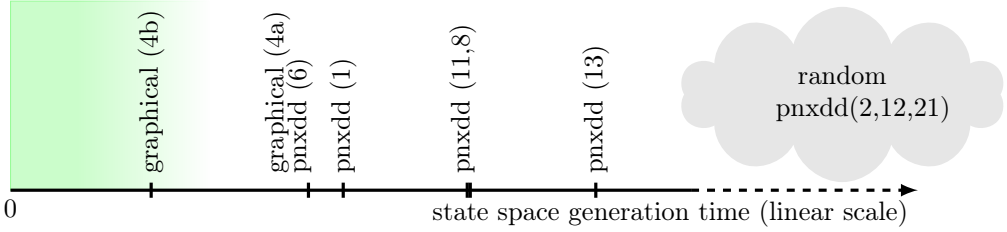
The last place is f. The variables that relate this place to d are used in the second part of the couple, whereas the variables that relate d to the places in its group are in the first part. So, f probably represents a shared data. We thus put place f in its own cluster, for all colors.

```
Clusters c, c1, c2, c3, c4, f;
Rules
  cluster of ● in e is c;
  cluster of ❶ in a,b,c is c1;
  cluster of ❷ in a,b,c is c2;
  cluster of ❸ in a,b,c is c3;
  cluster of ❹ in a,b,c is c4;
  cluster of ❶ in f is f;
  cluster of ❷ in f is f;
  cluster of ❸ in f is f;
  cluster of ❹ in f is f;
  cluster of {❶} × {❶,❷,❸,❹} in d is c1;
  cluster of {❷} × {❶,❷,❸,❹} in d is c2;
  cluster of {❸} × {❶,❷,❸,❹} in d is c3;
  cluster of {❹} × {❶,❷,❸,❹} in d is c4;
  c < c1 < c2 < c3 < c4 < f;
```

Listing 4: Clustering using alignment

Figure 4b shows the result of this analysis. Note that for each color ❶, ❷, ❸, or ❹, we create a cluster that contains all the highlighted places for this color only. For place d, each cluster contains all possible colors for the second part of the Cartesian product.

We compare both clusterings below. For this third model, the state space generation could not finish using some algorithms of PNXDD (2,12,21). Graphical clustering and ordering is still better than all algorithms implemented in PNXDD. As for other models, none of the random clusterings could finish.



6 When we fail at understanding the model

The Model Checking Contest of SUMo 2011 workshop uses seven models, three of which are Place/Transition Petri nets and the four others Symmetric Petri nets. In this article, we provide clustering for `Flexible Manufacturing System`, `Kanban` and `Shared Memory`. Two models are not discussed: `Philosophers` and `Token Ring`. The `Philosophers` model has been studied a lot for use with Decision Diagrams. Before writing this article, we already knew that the best order is when each philosopher and its left (or right) fork is put in its own cluster. Because of this knowledge, trying to find graphical information would be biased. The `Token Ring` model does not contain enough places (1) and transitions (2).

The two remaining models are `MAPK` (Figure 5) and `Peterson`'s algorithm (Figure 6). For them, we did not find how to give semantics to the graphical information. Note that these two models are considered as hard to understand by humans. This might be an explanation: because their modelers could not find a good disposition of places and transitions, we cannot give semantics to graphical information, and thus the models are hard to understand.

7 Discussion on the approach

We propose in this article to extract semantics from the graphical part of a Petri net. It works on some models, but does not work on some others. Moreover, all people do not always agree on how to identify the processes in the Petri net. This approach is thus fragile. We provide in this section some remarks on its applicability, and on other techniques that could be used along graphical information to improve clustering.

7.1 This approach depends on the school of modeling.

Depending on where we have studied, and where we work, we may not create models in the same way. The most obvious difference is putting aligned processes horizontally or vertically. But some people can also prefer to show processes using surfaces instead of alignments. We should investigate which representations are used, and where they are modeled.

A side effect is that every Petri net should define an author and the institutes where the author has studied and workse Using this information, giving semantics to graphical information could be enhanced. The "author" field is already present in the Petri Net Markup Language [10], but there is currently no traceability of the author's institutions, nor of the school of modeling for this Petri net.

7.2 Graphical information cannot be processed easily

The Petri Net Markup Language handles absolute positions for the Petri net elements (places, transitions, labels...). The problem with absolute positioning is that analysis of the graphical information is not easy. Some other positioning schemes exist. For instance, Ti*k*Z [11] provides a way to define a position relative to another one (above, below, ...). Such positions can be processed more easily.

7.3 Labels can also be used to detect components.

We use only graphical information on Place/Transition Petri nets in this article. For Symmetric Petri nets, we also use domains and variables on the arcs. Another interesting information to compute clustering is the labels of places and transitions. In Place/Transition Petri nets, they are often composed of a textual prefix, a textual suffix, and some numbers in the middle. These numbers correspond to colors in an equivalent colored Petri nets. Extracting the naming patterns of places can greatly enhance the clustering.

7.4 This approach may be redundant with graph analysis.

Ordering the Decision Diagrams used during the model checking of a Petri net has been explored for a long time. We can cite again [7], which is a survey of existing algorithms.

Instead of using the graphical information, they are usually based on the graph structure of the Petri net. These techniques, from the structural analysis field, can lead to very good results. Note that combining them with graphical information and naming analysis is often possible.

First, traps and siphons [12] are a way to detect components, and thus clusters. They can be computed efficiently, for instance in [13]. Invariants are another way to define clusters, but they show both processes and data. A good point is that they can even be computed in colored Petri net [14].

8 Conclusion

Throughout this article, we show Petri nets from the Model Checking Contest of SUMo 2011 workshop. For each one, we try to understand how their modelers put semantics in their graphical information. Whereas we cannot identify such information in some models (`MAPK` and `Peterson`'s algorithm), it provides really good results for the others.

We extract semantics from several kinds of graphical information: the alignment of places and transition, the graphical proximity of places and groups, and the surfaces delimited by arcs. This semantics is used to define clustering and ordering of the places and tokens.

For all these examples, we use the graphical information to improve symbolic model checking of the Petri net. To do so, we extract clustering and ordering from the graphical part of each model. Using them, we compare the time taken to generate the state space with several clustering and ordering algorithms implemented in `PNXDD`, and with fully random clusters and orders. In all models where graphical clustering could be defined, we obtain improvements over `PNXDD`. No random clusters and orders could run in the time and memory limits, which shows that the semantics extracted from graphical information makes sense.

At the beginning of this work, we were only hoping to obtain improvements over some of the algorithms in `PNXDD`. Prior the benchmarks, we did no selection of the models, and of the inferred graphical semantics. Thus, the good results seem promising.

From these examples, two conclusions arise: (1) in education, Petri nets are always introduced as a graphical formalism. In research, they have been deteriorated into a mathematical only formalism, by losing graphical information; (2) graphical information in Petri nets has semantics, but unclear one.

We believe that, contrary to their usual presentation in scientific publications, Petri nets are truly a graphical formalism, and their mathematical representation is only a projection.

Further work should identify more graphical semantics, and the cases in which they apply. We may not be able to create algorithms to extract always semantics from graphical information. But when it is possible, a study with fully automatic algorithms should check that the inferred semantics can be used with no user knowledge of the models.

References

1. Sun, Y., Gray, J., Langer, P., Wimmer, M., White, J.: A WYSIWYG Approach for Configuring Model Layout using Model Transformations. In: DSM'10: Workshop on Domain-Specific Modeling @ Splash. (2010)
2. SMV Group: Algebraic Petri Nets Analyzer (2010) http://alpina.unige.ch.
3. Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers **35**(8) (1986) 677–691
4. Couvreur, J.M., Encrenaz, E., Paviot-Adet, E., Poitrenaud, D., Wacrenier, P.A.: Data Decision Diagrams for Petri Net analysis. In: ICATPN '02: 23rd International Conference on Applications and Theory of Petri Nets. (2002) 101–120
5. Bollig, B., Wegener, I.: Improving the variable ordering of obdds is np-complete. IEEE Transactions on Computers **45**(9) (1996) 993–1002
6. Buchs, D., Hostettler, S., Marechal, A., Risoldi, M.: AlPiNA: A symbolic model checker. In: Petri Nets '10: International Conference on Theory and Applications of Petri nets. (2010) 287–296
7. Rice, M., Kulhari, S.: A survey of static variable ordering heuristics for efficient BDD/MDD construction. Technical report, UC riverside (2008)
8. Hong, S., Paviot-Adet, E., Kordon, F.: PNXDD Model Checkers – https://srcdev.lip6. fr/trac/research/neoppod/ https://srcdev.lip6. fr/trac/research/NEOPPOD/.
9. Hong, S., Kordon, F., Paviot-Adet, E., Evangelista, S.: Computing a Hierarchical Static Order for Decision Diagram-Based Representation from P/T Nets. ToPNoC: Transactions on Petri Nets and Other Models of Concurrency (submitted) (2010)
10. Hillah, L.M., Kindler, E., Kordon, F., Petrucci, L., Trèves, N.: A primer on the Petri Net Markup Language and ISO/IEC 15909-2. Petri Net Newsletter **76** (2009)
11. Tantau, T., Feuersaenger, C.: TikZ ist kein Zeichenprogramm http://www.ctan.org/tex-archive/graphics/pgf/.
12. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (1989) 541–580
13. Barkaoui, K., Lemaire, B.: An Effective Characterization of Minimal Deadlocks and Traps in Petri nets Based on Graph Theory. In: 10th Int. Conf. on Application and Theory of Petri Nets ICATPN'89. (January 1989) 1–21
14. Couvreur, J.M., Haddad, S., Peyre, J.F.: Computation of generative families of positive semi-flows in two types of coloured nets. International Conference on Theory and Applications of Petri Nets (1991)

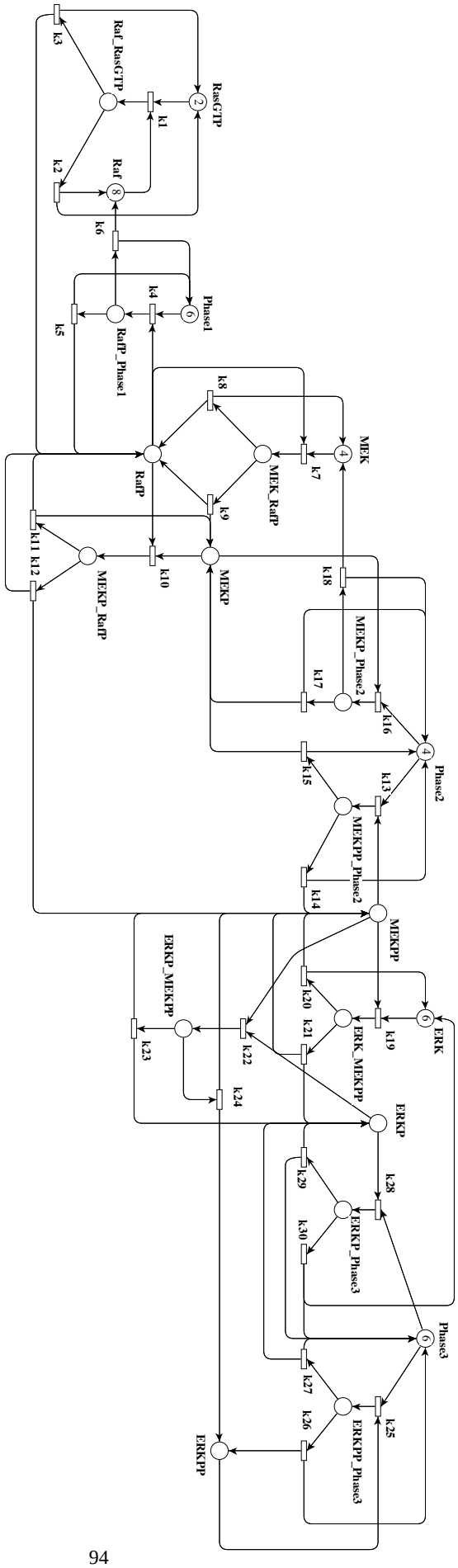


Fig. 5: MAPK: Mitogen-activated protein kinase kaskade

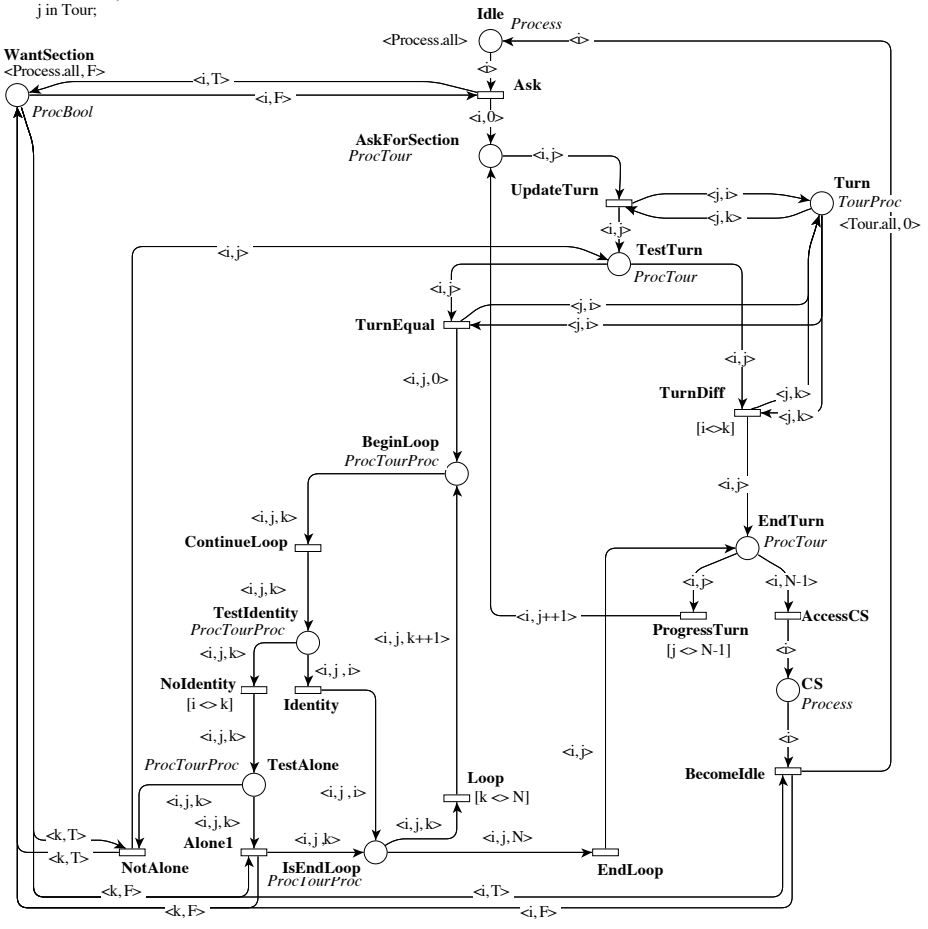


Fig. 6: `Peterson`'s algorithm