Event splitting:
the way to survive when regions fail

Jordi Cortadella

Department of Computer Science
Universitat Politècnica de Catalunya, Barcelona

Abstract. The theory of regions, originated from the work by Ehrenfeucht and Rozenberg, established the bridge between transition systems and Petri nets and a path towards the friendly visualization of concurrent behaviors. Unfortunately, not every transition system can be represented with a Petri net in which every event corresponds to a single transition. However, a Petri net can always be found if events can be split and represented by multiple transitions. When applying event splitting, an exponential space of solutions arises, each one delivering a different Petri net. Selecting one with gracious properties is a challenge and an open problem. This paper will informally illustrate the impact of event splitting using simple examples and will discuss directions of research for this problem.

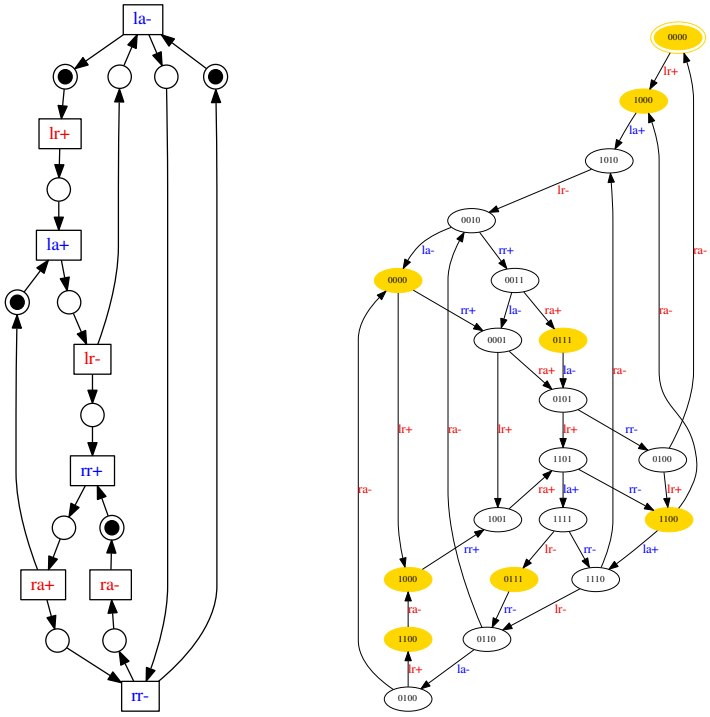
1 Motivation

Since I first met Alex Yakovlev in 1994, I have had the pleasure to share many unforgettable experiences in our professional and personal lives. Petri nets, asynchronous circuits and design automation have been permanent *leitmotifs* in our academic research. Along with our endearing colleagues, Mike Kishinevsky, Alex Kondratyev and Luciano Lavagno, we managed to solve challenging problems and many of them ended up enhancing the functionality of `petrify` [4], a tool that is still alive today.

During the nineties, our passion was to introduce automation in the design and verification of asynchronous circuits. We had a devotion for Petri nets and their interpretation as Signal Transition Graphs (STGs) [11]. Since logic synthesis techniques required state information with explicit values for the binary signals [5], we were often generating transition systems (TSs) from Petri nets. At that time, we decided to use Binary Decision Diagrams [2] for representing sets of states symbolically and handle the state explosion problem in a manageable way.

Figure 1 shows a classical synthesis example. Figure 1a depicts the specification of an asynchronous controller with an STG[1], whereas the corresponding TS is shown in Fig. 1b. Each state has an associated binary code with as many

[1] Most drawings in this paper have been automatically generated by `graphviz` [9], a tool set that can be found in `www.graphviz.org`.



(a) Signal Transition Graph. (b) TS with encoding conflicts.

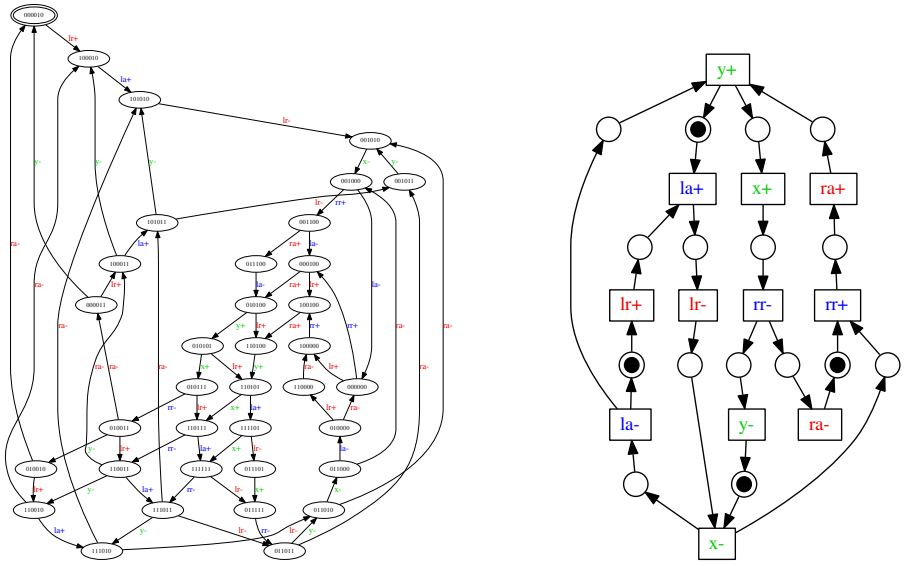
Fig. 1. Generation of a binary-encoded TS from an STG.

digits as signals in the system. The shadowed states represent encoding conflicts, i.e., multiple states sharing the same binary code.

The state encoding problem was one of the many problems we had to solve for the synthesis of asynchronous controllers [6]. We managed to find a satisfactory solution adding new signals and transforming the TS, as shown in Fig. 2a. In this particular example, two new internal signals, x and y, are introduced to disambiguate the state encoding conflicts.

We were often facing problems that required transformations at the level of TS: hiding signals, adding new state signals, reducing concurrency, etc. However, evaluating the impact of those transformations was a challenge, since the visualization and analysis of TSs was an extremely arduous task.

The TSs we had to manage often had hundreds or thousands of states, with a lot of concurrency embedded in their semantics. Unfortunately, we were no able of analysing the results of the transformations with a reasonable human effort. We were asking ourselves: why should not we be able to visualise the behaviour of a TS as an STG? It was then when the theory of regions came into play [8, 1] as the instrument to return to the Petri net world from a TS.



(a) TS without encoding conflicts. (b) STG with the new signals.

Fig. 2. Specification of the asynchronous controller after solving the encoding problem.

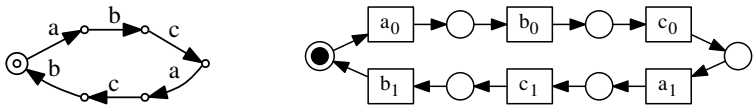
At that time, we were fascinated by the theory proposed by Ehrenfeucht and Rozenberg, that gave us the vehicle to synthesise STGs after manipulating our TSs. The implementation of that theory was the genesis of `petrify` [7]. With that vehicle we could generate the STG shown in Fig. 2b in which the causality relations of the new signals could be cleanly observed and analysed.

2 Petri net synthesis is not that simple

We immediately realised that things were not as simple as expected. For a TS to be representable as a Petri net, with each event represented by a single transition, a set of conditions must be fulfilled: the TS must be *elementary* [8]. In [7] we presented a more general concept, *excitation closure*, that defined a set of conditions to generate a Petri net with bisimilar behaviour [10]. A TS fulfilling those conditions is called an Excitation-Closed TS (ECTS). Unfortunately, most TSs are not ECTSs. A very simple example is shown in Fig. 3a.

However, the synthesis of a bisimilar Petri net is always possible by applying event splitting, i.e., allowing each event to be represented by more than one transition in the Petri net. For example, an event a could be represented by two transitions with labels a_0 and a_1 (the subindices represent different instances of the same label).

In the worst case, a degenerated solution consisting of a Petri net structurally isomorphic to the TS can be constructed: each place corresponds to one state

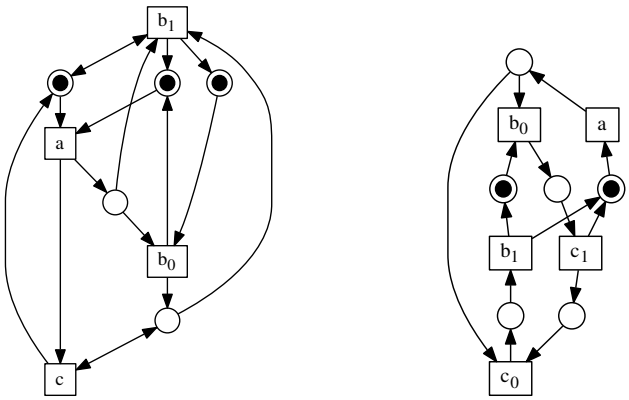


(a) Non-elementary TS. (b) Petri net obtained after splitting all events.

Fig. 3. Synthesis of a non-elementary transition system by event splitting.

of the TS and each transition to an arc. The initial state is represented by a marked place, as shown in Fig. 3b.

Unfortunately, this solution is quite uninteresting since it does not bring any additional information for the analysis of the system. The obvious question that comes to our mind is: can we minimise the number of events that need to be split to guarantee the synthesis of a Petri net?



(a) Splitting event b. (b) Splitting events b and c.

Fig. 4. Different solutions obtained after event splitting. Arcs with double arrow ($\bigcirc\leftrightarrow\square$) represent two arcs ($\bigcirc\rightleftarrows\square$).

Figure 4 depicts two bisimilar solutions obtained by splitting a different set of events. If we also consider the Petri net in Fig. 3b, we end up by having three different solutions with the following characteristics:

Figure	Places	Transitions	Arcs
3b	6	6	12
4a	5	4	18
4b	6	5	14

In this particular example, no solution with three transitions exists, since the original TS is not an ECTS. By analysing the previous table it is obvious to

realise that minimizing the number of transitions (event splits) is not always the best choice.

Even in the case of an ECTS, event splitting can be an option to generate a better visualisation of the behaviour. Figure 5 shows an ECTS with five events. A bisimilar Petri net with five transitions is depicted in Fig. 6a. The intricate relationship between places and transitions makes the analysis of this structure very tortuous. Instead, by simply splitting event a, the Petri in Fig. 6b is obtained, which clearly visualises the concurrency of events b and c and the choice between d and e.

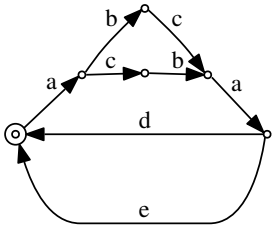
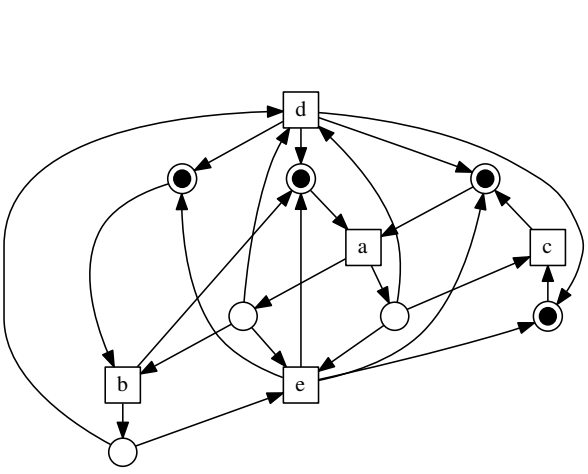
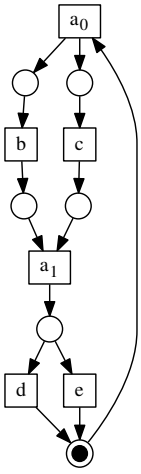


Fig. 5. Transition system.



(a) Without event splitting.



(b) Splitting event a.

Fig. 6. Two bisimilar Petri nets representing the behavior of the TS in Fig. 5.

3 Which events to split?

Event splitting is at the core of the Petri net synthesis problem. As we can suspect from the previous examples, the space of solutions can be exponential on the size of the TS. An essential question is the following:

> *How to measure the quality of a solution after selecting a set of events for splitting?*

As shown in Fig. 6, minimising the number of transitions may not always be the best choice, but doing a frenetic splitting may result in uninformative solutions. There is no clear answer for such question, but some directions for exploration are next discussed.

3.1 Minimising the number of transitions

Minimising the number of transitions is a strategy that may lead to more compact models. A possible approach could consist of finding a small set of events for splitting. This set would generate new regions to enforce the excitation closure for all events.

Figure 7a depicts a transition system and the set of minimal regions, $\{r_1, \ldots, r_6\}$, represented as shadowed sets of states. The excitation closure holds for all events except e. The only pre-region of e is r_4, but e is only enabled in one of the states.

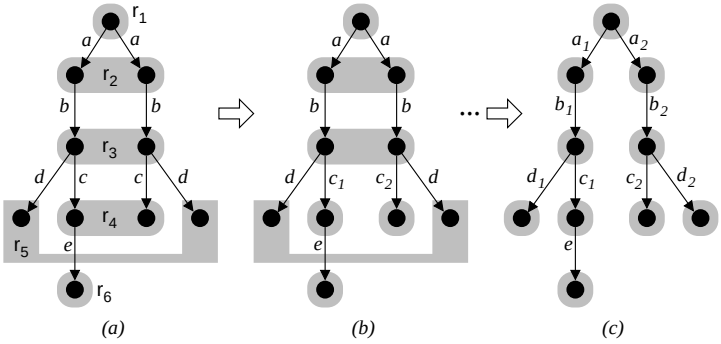


Fig. 7. The domino effect of greedy splitting (example from [3]).

A greedy approach, as proposed in [3], would consist in creating another region that would *separate* the two states in r_4. This could be achieved by splitting event c into c_1 and c_2 (Fig. 7b).

However, this myopic view solves the problem for e but creates another problem for c_1 and c_2. Applying the same strategy, excitation closure for c_1 and c_2 could be enforced by splitting b, etc, thus unleashing a domino effect that, in this particular case, would produce a complete event splitting, as shown in Fig. 7c.

This example shows that using naïve strategies for splitting may result in poor quality solutions.

An alternative approach could be proposed by globally analysing the excitation closure problem and resorting to the concept of state separation from the theory of regions [8]. This is illustrated in Fig. 8.

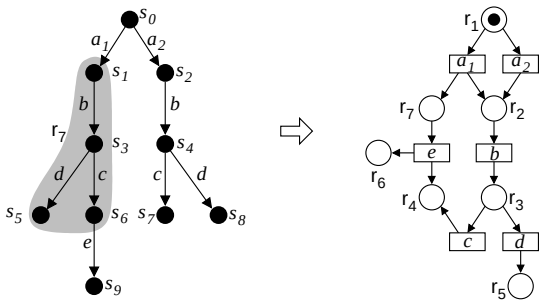


Fig. 8. New region (r_7) and Petri net synthesis after splitting event a.

By analysing the set of minimal regions $\{r_1, \ldots, r_6\}$ in Fig. 7a, we can calculate the pairs of non-bisimilar states that cannot be distinguished by the regions. Two states cannot be distinguished if there is no region such that one state is in the region and the other is not. In the example, the set of non-bisimilar pairs is:

$$\{(s_1, s_2), (s_3, s_4), (s_6, s_7)\}$$

Proposing heuristics to find set of states that *separate* these pairs is an interesting direction of research.

For example, the set $\{s_1, s_3, s_6\}$ could be a good candidate, but it would require two event splits to become a region: a and d. Instead, the set $\{s_1, s_3, s_5, s_6\}$ would guarantee the same separation with only one event split. This set corresponds to region r_7 in Fig. 8 and leads to the Petri net on the right, in which each place is annotated with the corresponding region.

3.2 Simplifying the structure of the Petri net

Having a nice visualisation contributes to giving a graphical intuition of the relationship between events. For example, producing series-parallel graphs or minimising the number of arc crossings in a picture is always a desired property for a Petri net.

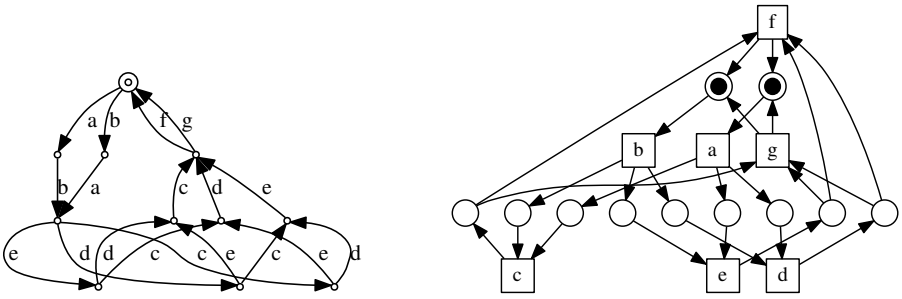
Analysing the connectivity and the causality relations between events can help to decompose sets of states in which an event is enabled. The TS in Fig. 5 is one example. Event a has two dispersed states in which the event is enabled. Additionally, each state is triggered by different sets of events ($\{b, c\}$ in one state and $\{d, e\}$ in the other state). The separation of enabling sets and the distinction

of trigger sets may be a criterion to split events. In this case, splitting event a results in the Petri net shown in Fig. 6b, which has nice structural and graphical properties.

3.3 An unexpected guest: τ

An alternative strategy to simplify the structure of a Petri net is to intentionally insert silent events (τ). In some cases, a new event can *collect* causality information between groups of events and contribute to better visualize their relationship.

An example is depicted in Fig. 9a, where two concurrent events (a and b) trigger another group of three concurrent events (c, d and e). This is represented by a two-way diamond preceding another three-way diamond. Similarly, c, d and e trigger two events in conflict (f and g) that later trigger events a and b.



(a) Transition system. (b) Petri net with bisimilar behaviour.

Fig. 9. TS with complex relationships between events.

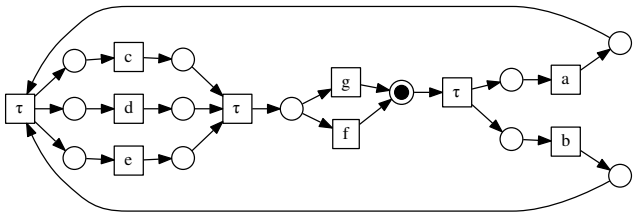


Fig. 10. Petri net after the insertion of τ events.

In a Petri net, a group of n concurrent events triggering another group of m concurrent events requires a set of $n \times m$ places representing the cross product

of relationships between pairs of events (see Fig. 9b). This situation can be analysed at the level of TS and insert silent events in strategic states that separate relationships between multiple events.

Figure 10 depicts a Petri net exhibiting the same behaviour as the one in Fig. 9b, but including some τ transitions. To be more precise, both Petri nets are weakly bisimilar [10]. The following table reports a comparative study about the structural properties of both Petri nets.

Figure	Places	Transitions	Arcs	Arcs/Nodes	Arc crossings
9b	11	7	27	1.50	12
10	12	10	26	1.18	0

For a fair comparison, both layouts have been generated by `graphviz`. The number of arc crossings is reported by the tool and gives an idea of the intricateness of the layout. Clearly, the layout shown in Fig. 10 is much more graphically intuitive. Quantitatively speaking, the ratio of arcs per node and the number of arc crossings are parameters highly related to the visualisation of the layout.

4 Surprise, surprise, ...

During a discussion on the problem of duplicate tasks in process mining, an interesting example came up. In process mining, the goal is to obtain a formal model from a set of behaviours represented by an event log.

Figure 11 shows a TS obtained by the event log at the left. Each one of the traces from the log corresponds to a different trajectory in the TS. The figure also shows a bisimilar Petri net after event splitting.

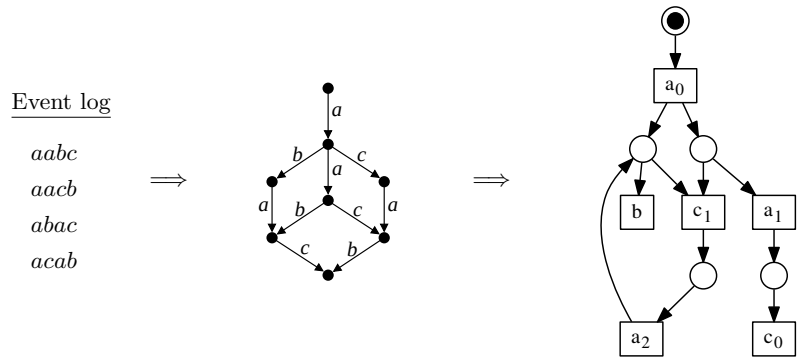


Fig. 11. TS and Petri net obtained from an event log.

It was interesting to realize that a much simpler Petri net, shown in Fig. 12, was able to generate exactly the same language. However, there is a small subtlety that differentiates them. The underlying TSs are not bisimilar, but they still are trace equivalent.

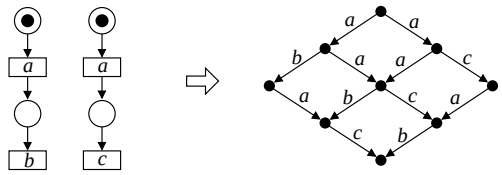


Fig. 12. Petri net generating the same language as the event log in Fig. 11.

The previous example suggests that event splitting can be solved in different ways depending on the equivalence that needs to be preserved during the synthesis of a Petri net.

5 Conclusions

For many years I have had the opportunity to share many exciting discussions about a large variety of research problems with Alex Yakovlev. After such a long time, some of the problems we have tackled have been solved, many of them are still open and many others are still unknown.

This paper just showed one of the open problems that will surely draw the attention of some researchers in the near future. My only hope is that the examples shown in the paper stimulate new ideas and discussions such as the one we had with Alex in Dresden (March 2016) when chatting about this topic and, more in particular, about the last example shown in the previous section.

References

1. Eric Badouel, Luca Bernardinello, and Philippe Darondeau. *Petri Net Synthesis*. Springer-Verlag, 2015.
2. Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, August 1986.
3. Josep Carmona. The label splitting problem. *Trans. Petri Nets and Other Models of Concurrency*, 6:1–23, 2012.
4. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, March 1997.
5. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer-Verlag, 2002.
6. Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alexandre Yakovlev. A region-based theory for state assignment in speed-independent circuits. *IEEE Transactions on Computer-Aided Design*, 16(8):793–812, August 1997.
7. Jordi Cortadella, Michael Kishinevsky, Luciano Lavagno, and Alexandre Yakovlev. Deriving Petri nets from finite transition systems. *IEEE Transactions on Computers*, 47(8):859–882, August 1998.

8. Andrzej Ehrenfeucht and Grzegorz Rozenberg. Partial (set) 2-structures, parts i-ii. *Acta Informatica*, 27:315–368, 1990.
9. John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz - Open Source Graph Drawing Tools. *Graph Drawing*, pages 483–484, 2001.
10. Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
11. Leonid Ya. Rosenblum and Alexandre Yakovlev. Signal graphs: From self-timed to timed ones. In *International Workshop on Timed Petri Nets*, pages 199–206, Washington, DC, USA, 1985. IEEE Computer Society.