Genet: a Tool for the Synthesis and Mining of Petri Nets (Tool paper)

J. Carmona
Universitat Politècnica de Catalunya
Barcelona, Spain

J. Cortadella
Universitat Politècnica de Catalunya
Barcelona, Spain

M. Kishinevsky
Intel Corporation
Hillsboro, USA

Abstract

State-based representations of concurrent systems suffer from the well known state explosion problem. *In contrast, Petri nets are good models for this type of systems both in terms of complexity of the analysis and in visualization of the model. In this paper we present* `Genet`*, a tool that allows the derivation of a general Petri net from a state-based representation of a system. The tool supports two modes of operation:* synthesis *and* mining. *Applications of these two modes range from* synthesis of digital systems *to* Business Intelligence.

1 Introduction

In this paper we present `Genet`, a tool that transforms a state-based representation of a system (a transition system [3]) into an event-based model (a Petri net [10]). Petri nets represent the concurrency explicitly and therefore are very good visualization objects for concurrent systems. Moreover, the complexity of the analysis techniques can be significantly alleviated if done at the level of the Petri net. Recently, some techniques have been presented for the problem of *Process Mining* [12]: given a set of traces, a process model (usually a Petri net) must be obtained to cover the traces seen so far and maybe more. The idea behind that is to capture (discover) the behavior that can only be seen by monitoring a real system.

The input of `Genet` is a finite transition system TS modelling a concurrent system, and the output is a weighted and bounded Petri net PN that has a certain relation with the TS. This relation is: $\mathsf{TS} \approx RG(\mathsf{PN})$ (synthesis mode [7]), or $L(\mathsf{TS}) \subseteq L(\mathsf{PN})$ (mining mode [6]), were $\approx$ denotes *observational equivalence* [3], $RG(\mathsf{PN})$ is the reachability graph of the Petri net, and $L(\mathsf{TS})$ ($L(\mathsf{PN})$) is the language of TS (PN).

Synthesis and mining might derive very different PNs. Let us use the example of Figure 1 to illustrate this: the TS depicted in 1(a) satisfies the synthesis conditions (see [7]) and therefore it can be synthesized or mined into a 4-bounded PN, shown in 1(b). Now imagine that the labels for arcs $200 \xrightarrow{a} 120$ and $004 \xrightarrow{c} 102$ are interchanged resulting in TS'. With this change, synthesis conditions from [7] do not hold and then there is no unique-labelled PN with reachability graph bisimilar to TS'. However, if mining is used, the PN of Figure 1(c) is obtained. Notice that some traces not present in TS' are possible in this PN, e.g. $(ba)^*$. In Process Mining applications, this overapproximation can be sometimes acceptable because the transition system represents only part of the possible system behavior.

Related work on synthesis and mining based on regions of languages has been published recently [5, 14], and some comparisons with the first reference can be found in [6]. A well-known tool for the synthesis of Petri nets is `petrify` [8]. `Genet` can be considered as an extension of `petrify` in the sense that the theory behind is a generalization of the one in `petrify`: first `Genet` can synthesize k-bounded Petri nets, whereas `petrify` generates 1-bounded (safe) Petri nets. Second, `Genet` can additionally do mining of k-bounded Petri nets. Hence data structures and algorithms behind `Genet` have been designed to cover the general case, thus being less optimized for the case of safe Petri nets.

2 Overview of the Theory Behind `Genet`

`Genet` is based on the theory of regions [9]. Intuitively, a *region* is a set of states in a transition system that has a homogeneous relation with respect to the events, i.e. every event either enters this set, or exits it, or never crosses its boundary. It corresponds to a place in the derived Petri net. In `Genet`, the notion of general region is used, were the corresponding place can have at most k tokens. Let us use the example of Figure 1 to illustrate the theory. The method assume that a k is initially given for the search of a k-bounded Petri net. The basic idea is that regions are represented by multisets (i.e., a state might have multiplicity greater than one). Figure 1(a) depicts a TS with 9 states and 3 events. After synthesis, the Petri net at Figure 1(b) is obtained. For illustration purpose we label each state with a 3-digit label that corresponds to the marking of places p_1, p_2 and p_3 of the Petri net, in the corresponding markings of

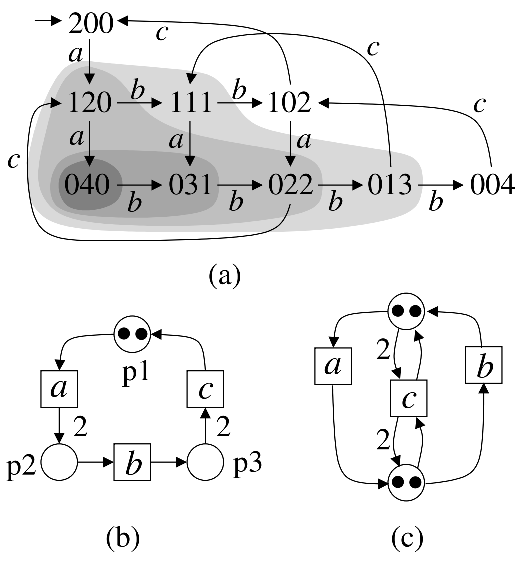

Figure 1. (a) A transition system, (b) an equivalent bounded Petri net, (c) Petri net mined from (a) where labels for arcs $200 \xrightarrow{a} 120$ and $004 \xrightarrow{c} 102$ are interchanged.

the Petri net reachability graph. The shadowed states represent the general region that characterizes place p_2. Each grey tone represents a different multiplicity of the state (4 for the darkest and 1 for the lightest). Each event has a constant gradient with respect to this region (+2 for a, -1 for b and 0 for c). The gradient indicates how the event changes the multiplicity of the state after firing. Gradients are used to weight the arcs between places and transitions in the derived Petri net: the gradient +2 for event a in the region shown corresponds to the arc with weight 2 between the transition a and the place p_2, meaning that every time the transition a fires it puts two tokens into p_2. Refer to [7] for further details on the theory.

3 Features of the Tool

Genet is a tool implemented in C++, and uses the following libraries: Cudd [4] (University of Colorado) to symbolically represent transition systems and regions, STL for data structures and algorithms, and PCCTS [11] for input parsing. The tool has the following features:

Symbolic representation: Binary Decision Diagrams (BDD) are used to represent the states of the transition system, its (disjunctive) transition relation, and the regions and intermidiate multisets found in the exploration. The operations on regions can be performed at the symbolic level. A symbolic algorithm to compute the regions has been implemented [7]. Different encoding schemes for representing a state are considered (see the *-enc* option of the tool).

Regions bound: Genet can be guided to search for regions in a given range. See the *-k* and *-min* options of the tool.

Projections: The tool can project the behavior of the transition system onto a given set of events before deriving the Petri net. This projection is done in two steps: first, events not included in the list of visible events are relabelled as silent events, and second, the fusion of states connected through silent events is iteratively applied. See the option *-prj* of the tool.

In the synthesis mode, the following features have been implemented:

Removal of redundant regions: Regions that are not necessary for the synthesis conditions are removed. These redundant regions correspond to *redundant* places in the corresponding Petri net.

Event splitting: When the synthesis conditions do not hold for the maximal bound allowed, the splitting of some events is automatically applied. Some heuristics have been proposed to this extent [7].

In the mining mode, the transition system represents a set of traces (called *event log*) of a system. The following features have been implemented:

Upper-bound for the covering: The user can indicate an upper bound for the number of regions allowed for covering an event. This may improve the visualization of the Petri net, but can loosen the overapproximation of the input TS by including more unspecified traces. See the option *-cov*.

Marked graph mining: The tool can be requested to mine *marked graphs*, a particular class of Petri nets where neither choice nor merge behavior is allowed. See the option *-mg*.

4 Experiments

Tables 1 report the results with the current version. Large examples have been added with respect to the tables in [6,7], to show the tool capacity. For each benchmark, the size of the transition system (states and arcs), number of places and transitions of the derived Petri net, and cpu time is shown.

The examples for the table on the left are syntactic examples representing typical behavior that can be nicely represented in a Petri net: a system with shared resources (ShRes), a producer-consumer environment (ProdCons), and a pipeline of n processes (BoundPipe). Hence in this table we show how Petri nets can be synthesized from the underlying behavior of these benchmarks. Intuitively, both the bound of the derived net and the size of the transition system are the main factors that define the complexity of the underlying algorithms.

The benchmarks on the right are real logs and are used for testing the mining mode of Genet. It is interesting to note the succinctness of the derived Petri nets: for the last example, a state space of more than five thousand states is summarized in a Petri net with eighteen places.

benchmark	$\|S\|$	$\|E\|$	$\|P\|$	$\|T\|$	cpu
SHRES(3,2)	63	186	13	12	0s
SHRES(4,2)	243	936	17	16	0s
SHRES(5,2)	918	4320	24	20	0s
SHRES(4,3)	255	1016	17	16	0s
SHRES(6,4)	4077	24372	25	24	18s
SHRES(7,5)	16362	114408	29	28	25m
PRODCONS(3,2)	24	68	8	7	0s
PRODCONS(4,2)	48	176	10	9	0s
PRODCONS(3,3)	32	92	8	7	0s
PRODCONS(4,3)	64	240	10	9	0s
PRODCONS(6,3)	256	1408	14	13	0s
PRODCONS(8,3)	1024	7424	18	17	2s
PRODCONS(8,5)	1536	11520	18	17	1h10m
BOUNDPIPE(4)	81	135	8	5	0s
BOUNDPIPE(5)	243	459	10	6	1s
BOUNDPIPE(6)	729	1539	12	7	6s
BOUNDPIPE(7)	2187	5103	14	8	48s
BOUNDPIPE(8)	6561	16767	16	9	12m
BOUNDPIPE(9)	19683	54675	18	10	1h50m

benchmark	$\|S\|$	$\|E\|$	$\|P\|$	$\|S\|$	cpu
groupedFollowsa7	18	7	7	11	0s
groupedFollowsal1	15	7	12	15	0s
groupedFollowsal2	25	25	15	25	0s
herbstFig6p21	16	7	11	16	0s
herbstFig6p34	32	12	18	32	0s
herbstFig6p41	20	14	16	18	0s
staffware_15	31	19	19	31	0s
pn_ex_10	233	11	16	145	0s
a12f0n50_1	78	77	17	80	0s
a12f0n50_2	151	150	21	92	0s
a12f0n50_3	188	187	21	92	0.5s
a22f0n00_1	1209	1208	16	78	9m
a22f0n00_2	3380	3379	16	78	15m
a22f0n00_3	5334	5333	16	78	32m

Table 1. Synthesis of parameterized benchmarks (left), and mining of event logs obtained from [2] (right).

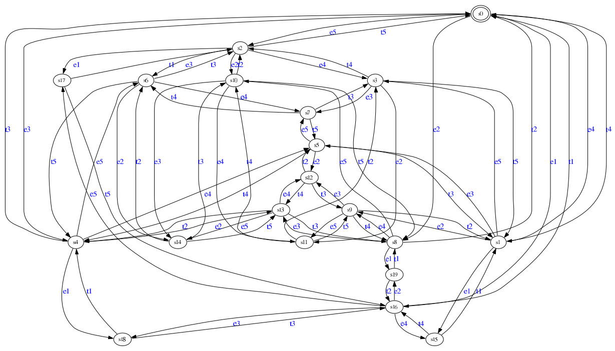

Figure 2. Shared resources system: transition system.

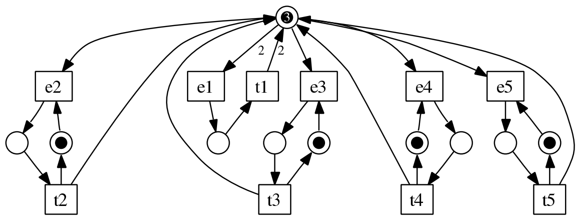

Figure 3. Shared resources system: Petri net.

5 Examples on Using the Tool

Examples extracted from real examples or event logs are presented. Figures 2 and 3 show a transition system and its corresponding 3-bounded Petri net, synthesized by `Genet`. Figure 4 shows how the tool can be used to project part of the system behavior into a set of events. In the figure, only the behavior regarding processes $P_2 \ldots P_5$ is considered.

Figure 5 shows the mining of a complex system. Also, Figure 6 shows a situation where mining derives a better visualization, on the event log represented by the transition system of Figure 6(a).

6 Tool Availability, Web Page and Tutorial

The tool has been developed and tested under Linux, and tested in Solaris. It is for the time being text-based. However, Petri nets and transitions systems can be graphically drawed with the public domain `draw_astg` tool, available at [1]. Although the tool is evolving to incorporate extensions of the existing theory, there is a stable core that has been reached after several improvements of the initial prototype of the tool presented in [7]. Some of these improvements led, for instance, to a very efficient exploration

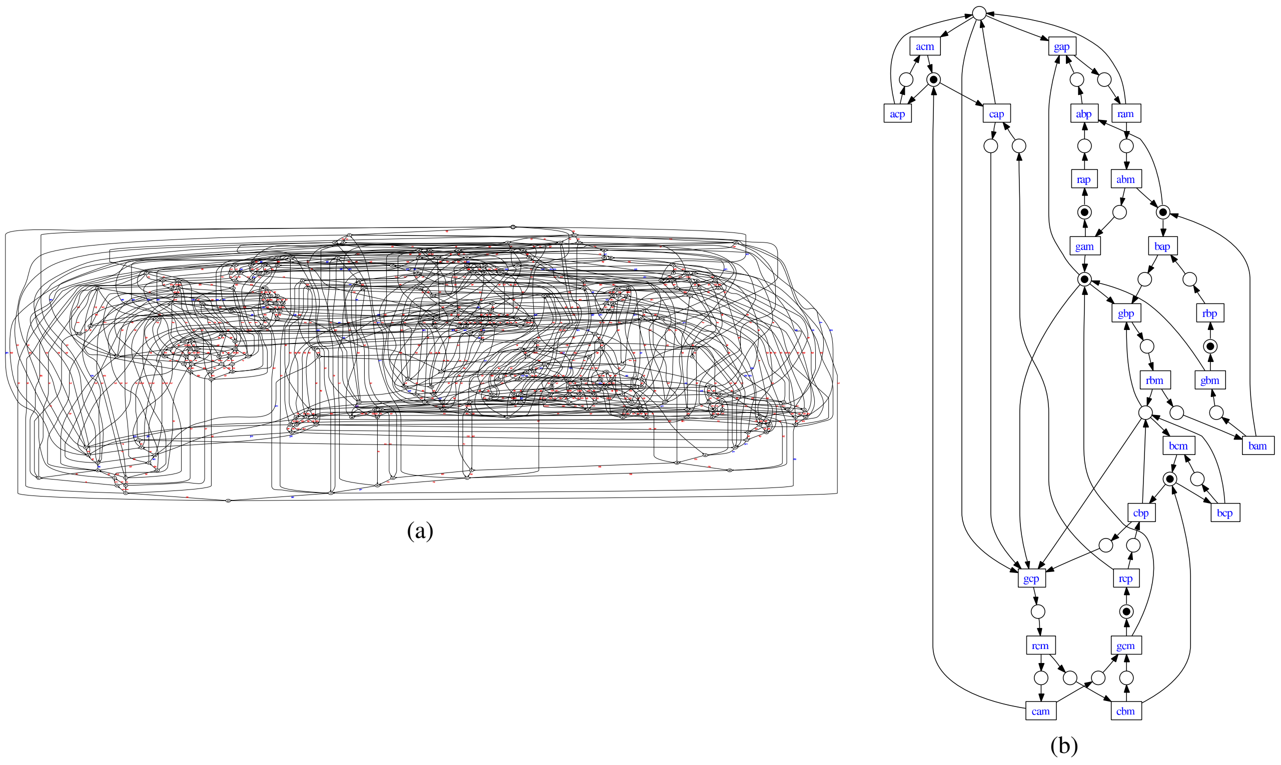

Figure 5. (a) Transition system, (b) Mined Petri net.

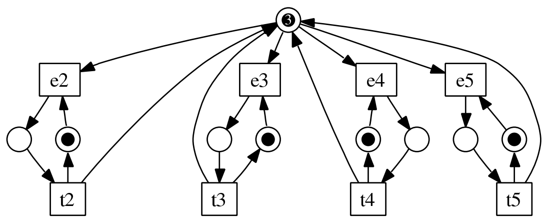

Figure 4. Synthesized Petri net for a subset of a system (processes $P_2 \ldots P_5$).

of the region space. Moreover, as a tool for Process Mining, `Genet` is in the process of being incorporated into the ProM framework [13]. There is a web page for the tool:

http://www.lsi.upc.edu/∼jcarmona/genet.html

were related papers, a tutorial and the Linux/Solaris binaries can be obtained.

7 Conclusions and future work

`Genet` is a tool to support the synthesis and mining of concurrent systems. We foresee several applications of the tool in areas like knowledge discovery and synthesis of digital circuits, among others. Our current work is focused into incorporating clustering techniques to automatically select parts of the system that can be composed afterwards and therefore *views* of the initial system can be produced.

Acknowledgements

We would like to thank Stanislavs Golubcovs for providing the example of Figure 5. This work has been supported by the project FORMALISM (TIN2007-66523), and a grant by Intel Corporation.

References

[1] Petrify. http://www.lsi.upc.edu/∼jordicf/petrify/.

[2] Process mining. www.processmining.org.

[3] A. Arnold. *Finite Transition Systems*. Prentice Hall, 1994.

[4] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In M. R. Light-

ner and J. A. G. Jess, editors, *ICCAD*, pages 188–191. IEEE Computer Society, 1993.

[5] R. Bergenthum, J. Desel, R. Lorenz, and S.Mauser. Process mining based on regions of languages. In *Proc. 5th Int. Conf. on Business Process Management*, pages 375–383, Sept. 2007.

[6] J. Carmona, J. Cortadella, and M. Kishinevsky. A region-based algorithm for discovering Petri nets from event logs. In M. Dumas, M. Reichert, and M. C. Shan, editors, *BPM*, volume 5240 of *LNCS*, pages 358–373. Springer, 2008.

[7] J. Carmona, J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. A symbolic algorithm for the synthesis of bounded Petri nets. In *29th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency*, June 2008.

[8] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Trans. on Information and Systems*, E80-D(3):315–325, Mar. 1997.

[9] A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures. Part I, II. *Acta Informatica*, 27:315–368, 1990.

[10] T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, pages 541–580, Apr. 1989.

[11] T. J. Parr. *Language Translation Using PCCTS and C++*. 1995.

[12] W. M. P. van der Aalst and C. W. Günther. Finding structure in unstructured processes: The case for process mining. In T. Basten, G. Juhás, and S. K. Shukla, editors, *ACSD*, pages 3–12. IEEE Computer Society, 2007.

[13] W. M. P. van der Aalst, B. F. van Dongen, C. W. Günther, R. S. Mans, A. K. A. de Medeiros, A. Rozinat, V. Rubin, M. Song, H. M. W. E. Verbeek, and A. J. M. M. Weijters. ProM 4.0: Comprehensive support for *eal* process analysis. In J. Kleijn and A. Yakovlev, editors, *ICATPN*, volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer, 2007.

[14] J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. In *Petri Nets*, pages 368–387, 2008.

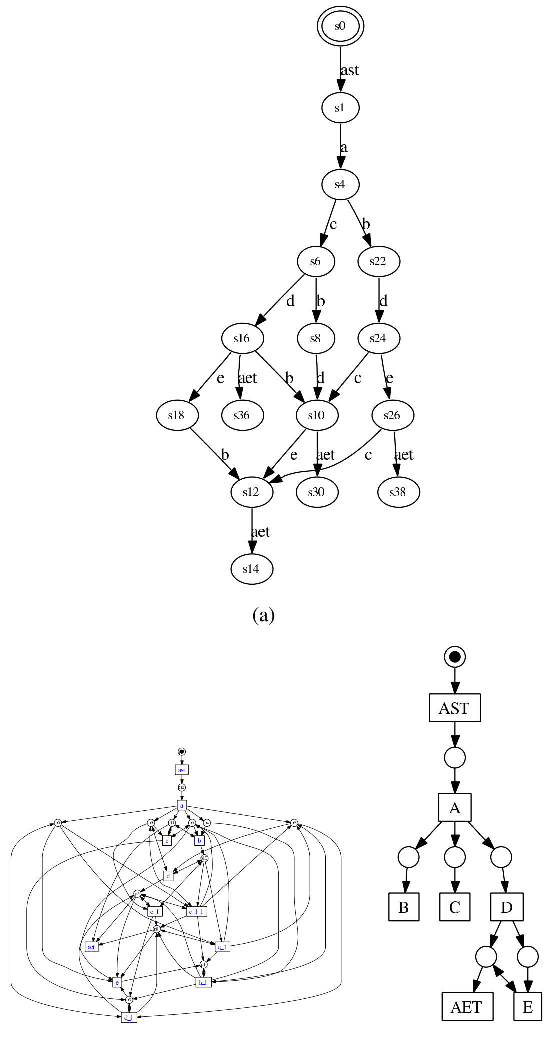

Figure 6. (a) Transition system specifying the initial log, (b) Synthesized Petri net from the transition system of Figure 6(a), (c) Mined Petri net from the transition system of Figure 6(a).