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: \( \text{TS} \approx RG(PN) \) (synthesis mode [7]), or \( L(\text{TS}) \subseteq L(PN) \) (mining mode [6]), where \( \approx \) denotes observational equivalence [3]. \( RG(PN) \) is the reachability graph of the Petri net, and \( L(\text{TS}) \) (\( L(PN) \)) is the language of \( \text{TS} \) (\( PN \)).

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 \( 200 \rightarrow 120 \) and \( 004 \rightarrow 102 \) are interchanged resulting in \( \text{TS}' \). With this change, synthesis conditions from [7] do not hold and then there is no unique-labelled PN with reachability graph bisimilar to \( \text{TS}' \). However, if mining is used, the PN of Figure 1(c) is obtained. Notice that some traces not present in \( \text{TS}' \) are possible in this PN, e.g. \( (bu)^\ast \).

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

Figure 3. Shared resources system: Petri net.

Figure 2. Shared resources system: transition system.

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 drewed with the public domain drawastg 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...
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 Golubcoves 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


