Leiden Universiteit

Bachelor's Thesis
to achieve the university degree of
Bachelor Computer Science

Petri nets & Property Directed Reachability

Olaf Randel

Supervisors:

Dr. A.W. Laarman
Dr. L. Cao

August 16, 2019

Abstract

Model checking is the practice of automatically deciding whether a system adheres to a certain specification, in order to prevent bugs. Model checking is therefore more rigorous than testing, since it also proves the absence of bugs.
Property Directed Reachability (PDR) is a method for model checking, in which the model is represented as propositional logic and the reachability of a state within that model is derived through incrementally constituting an invariant using satisfyability solvers.
Petri nets are a modeling language for describing parallel systems, which are systems in which different processes can progress concurrently. The PDR method has not yet been tried for Petri nets. The challenge in implementing this lies in the representation of Petri nets in symbolic logic.
In this thesis we illustrate how to use the PDR algorithm to check Petri net models. We do this by encoding the Petri net as a Boolean function, which serves as input for the symbolic PDR algorithm.
We implemented this encoder in Java, allowing us to test the efficiency of the method against competing model checking tools. Initial experimentation shows that the PDR method is comparatively inefficient.

Contents

1 Introduction

1.1 Relevance

Model checking is a form of system verification. A way to confirm whether a system conforms to certain design specifications. This is done in order to prove the design has no bugs.

Christel Baier and Joost-Poeter Katoen describe model checking as

> ” an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model.”

in their book ’principles of model checking’, [1].

A Petri net is a mathematical modeling language, which have become a prominent model for expressing parallel systems.
With the Petri net model being so prominent, and model checking being so vital, push-button verification is in high demand. The Model Checking competition, which posits a large set of Petri net problems for competing model checking tools, is founded on this demand. Before this research, a tool based on the Property Directed Reachability method had not yet been developed.

1.2 Research question

Property Directed Reachability could prove to be a method highly suitable to the analysis of Petri nets, due to the locality of the changes they undergo.

Since the method derives the reachability of a state from induction over possible changes, the fact that the transitions in a Petri net affect only a select group of connected elements could indicate the this method is highly suitable for application on Petri nets. The combination of the two is thus a worthwhile avenue to explore for possible innovation. However, the way to implement this method is not self-evident. The research question is therefore:

How can we prove the correctness of a Petri net using property directed reachability?

1.3 Approach

In order to simplify the problem to the scope of the project, we have to find a way to encode Petri nets as logical propositions. In this work we constrain ourselves only to something called "1-safe Petri nets". This term is explained in Chapter 2. To explain our method for encoding any 1-safe Petri net as a logical proposition, we first use an example Petri net to go by. This example has a known solution, allowing us to validate our encoding by running the Z3 PDR implementation [4].

1.3.1 Example

The example Petri net models the following scenario: A ferryman intends to transport a wolf, a goat and a cabbage over a river. His vessel can contain himself and up to 1 of these items at a time. The ferryman is incapable of leaving the goat alone with either of the other items on the shore while the he is on the river, since the goat would eat the cabbage, or the wolf would eat the goat. The only way for the ferryman to complete his assignment is by taking the goat back with him after transporting the second item to the opposing shore and then shipping the goat towards the destination again as his last action. The entities and possible actions are illustrated in Figure 1.1.
The four entities all start on the left shore. They can take up one position at

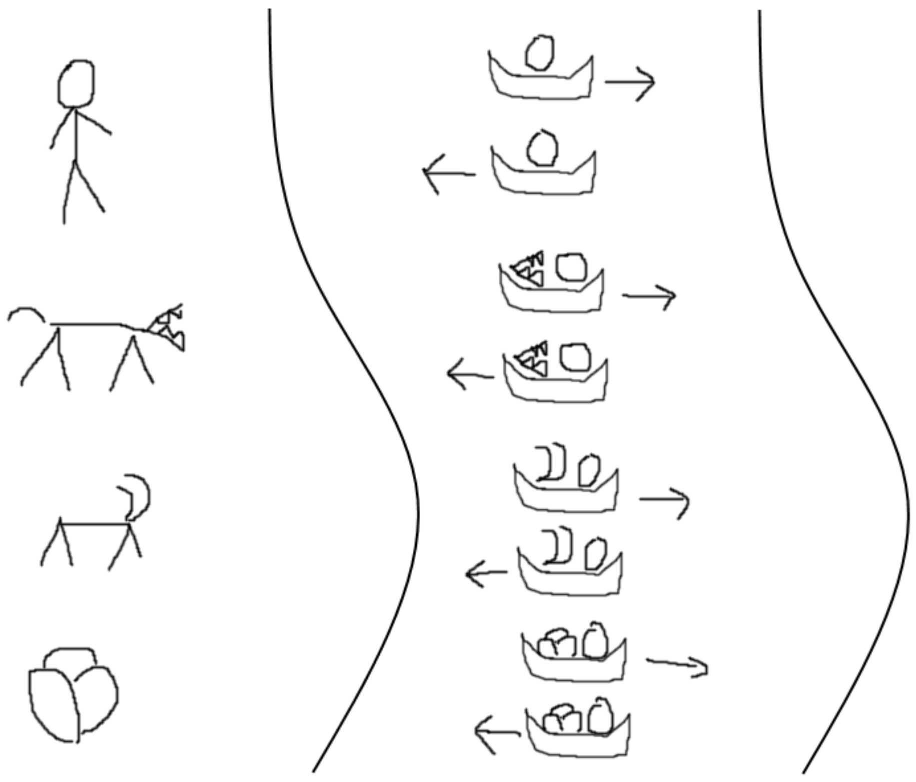

Figure 1.1: The ferryman scenario.

a time, either on the left shore or on the right. The ferryman can complete one of eight possible actions at a time, crossing the river with one of the other entities that is on the same shore. How this example can be encoded as a Petri net is further elaborated in Chapter 2.

1.3.2 Solution

We by creating an encoding of this example we obtain a template to model an encoding algorithm by.
The java method "test" in the github repository `https://github.com/OlafRandel/Z3_petrinet` is the encoding of this example. It does indeed give the solution to the riddle.

1.4 Overview

Chapter 2 explains terms used in this thesis, and lists information on both Petri nets and the Property Directed Reachability method for model checking. Chapter 3 outlines the method used to state Petri nets in predicate logic. And finally, chapter 4 will show the outcome of the experiments using the developed encoder, and how they compare to those of other another method.

2 Background

2.1 Petri nets

A Petri net is a triple. $N = (P, T, A)$, P is the set of places, T of transitions, and A of arcs. Places and Transitions are *disjoint*, meaning there is no element that is both. Arcs are defined as $A \subseteq (P \times T) \cup (T \times P)$. Meaning as an arrow that either points from a place to a transition, or from a transition to a place [2]. A Petri net models a concurrent system. Places contain 'tokens', representing resources in the system that is being modeled. Any transition can be 'fired', if the places with arcs pointing from them to it contain a token. When a transition 'fires' it uses up the tokens in each of these places, and depositing a token in the places its outgoing arcs point to. The "firing" of a transition is an atomic step, evolving the system from one discrete state to the next. See Figure 2.1 for an example of a Petri net transition.

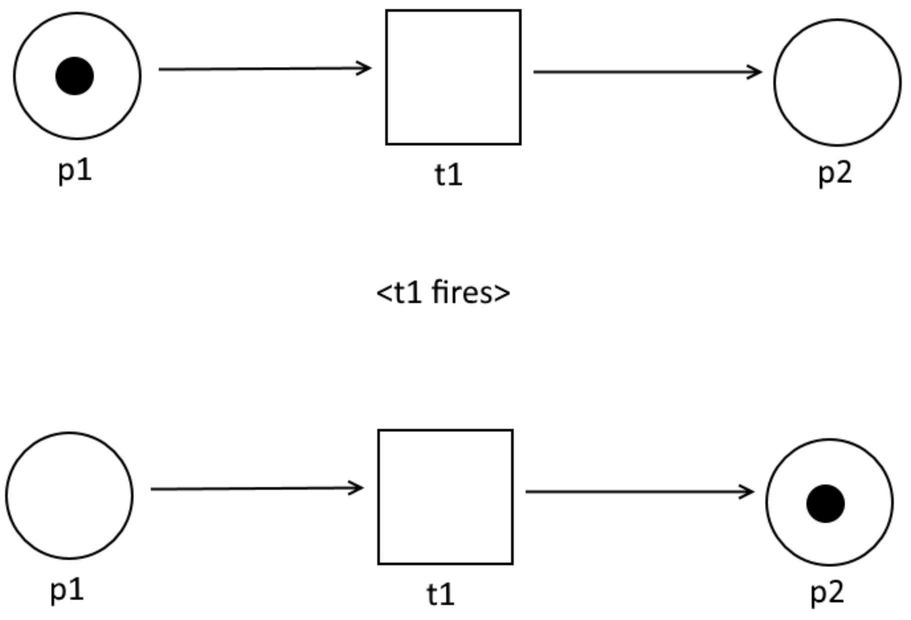

Figure 2.1: A transition firing.

The "marking" of a Petri net is the current distribution of tokens in each of the places. Each marking is a unique state. A 1-safe Petri net is one in which there's no marking where there are multiple tokens in one place. For each Petri net a graph can be drawn representing what markings are reachable from what markings through transitions. An example is given in Figure 2.2.

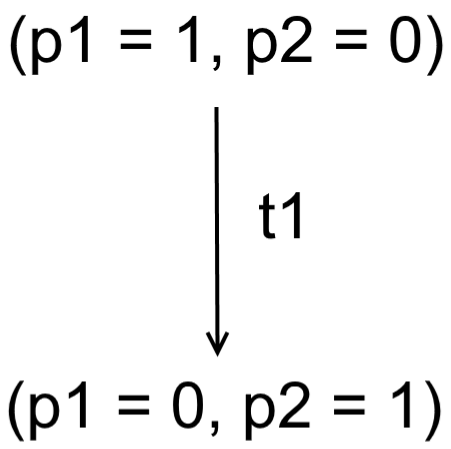

Figure 2.2: A reachability graph.

Semantically, a Petri net (P,T,A) is interpreted as a graph (m_0,M,E). The set of all possible markings is denoted with the capital letter M. The individual possible markings are number lower case m's, m_0 to m_n, such that $m_i \in M$. The first marking is m_0, the initial state. The set of possible changes in the markings can be represented as E, such that $E \subseteq M \times M$.
A popular format for Petri nets is the Petri net markup language (pnml), a program that uses Petri nets is best designed utilizing this language for it.

2.2 Symbolic Reasoning

Symbolic reasoning is a way to handle broad meanings in short sentences. For example, when a model contains two Boolean variables, equation 2.1 stands for a single valuation.

$$(p_1 \wedge \neg p_2) \tag{2.1}$$

But a broader set of satisfying solutions is shown by equation 2.2.

$$(p_1) \tag{2.2}$$

By not assigning a value to p2 we define the formula as correct for both the valuation $(p1 \land p2)$ and the valuation $(p_1 \land \neg p_2)$. It stands for two satisfying valuations. This allows for the reasoning over sets of valuations at once, rather than over each valuation individually. This allows fast computation over systems where changes happen mostly locally, between just a few variables, since much of the formula stays unchanged.

2.3 PDR

Model checking is any automated technique that checks whether a formal property always holds in a finite-state model. In other words, this property must be proven to be invariant. PDR is an algorithm that uses symbolic reasoning for model checking. In this algorithm the property (ϕ) that must be proven invariant is strengthened until it is inductive. This means that $E(\psi) \subseteq \psi$, where $\psi \Leftrightarrow \phi \cap X$ for some $X \subseteq M$ and E equals the image function of the transition relation. In other words, there is a set of states (ψ) for the model that are acceptable, and once the model is in a state in this set, it can not change to a state that is not in this set.
What's relevant for this project is that PDR can only work on a predefined model, encoded in symbolic logic. It requires an initial state (I), a transition relation (TR) and the supposed invariant (ϕ).

2.4 A Petri net example

A Petri net such as shown in Figure 2.3 could represent our ferryman scenario.

For every entity in the scenario two places in the Petri net are modeled, one for each possible location that entity can be in. In the initial position, the odd-numbered place in each pair of places is filled, denoting the fact that all entities are on the left shore. Every transition stands for an action the ferryman can take, T_1 can fire if the ferryman is on the left shore and will take him from there and deposit him in the right shore, T_2 does the opposite. Encoding this example into Boolean expressions to be used by the

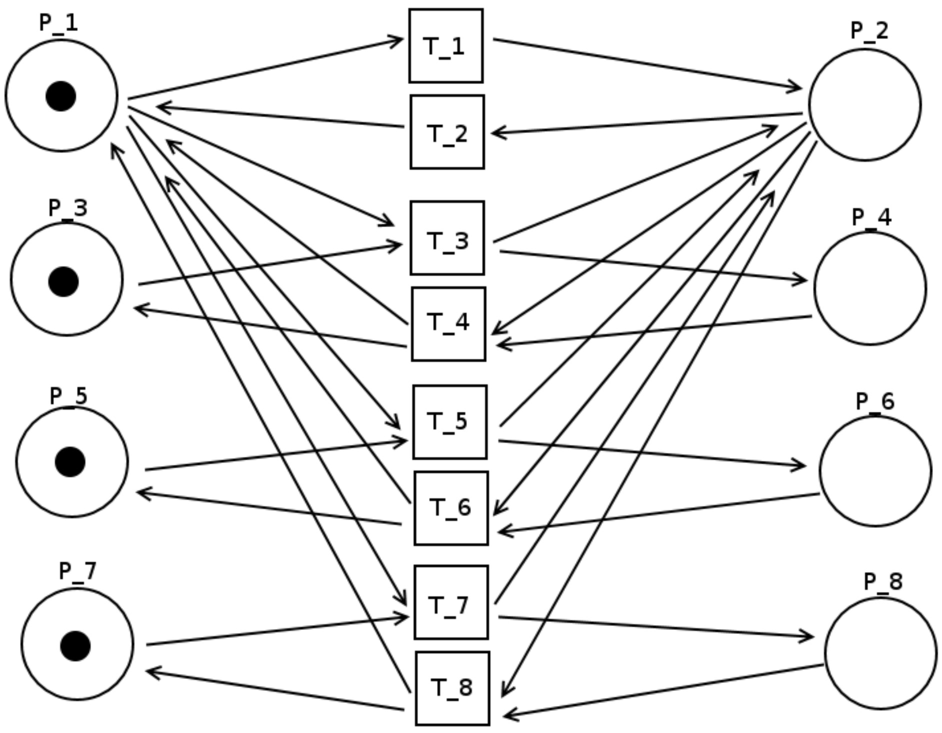

Figure 2.3: The ferryman scenario as a Petri net.

Z3 PDR implementation [4], should give a guideline to how such encoding can be done automatically. The example is demonstrated as correct when it returns the answer to the riddle. When applying the PDR method to this model, we must define the initial state (I) by given exact values for each of the eight places.

$$I := P_1 \wedge \neg P_2 \wedge P_3 \wedge \neg P_4 \wedge P_5 \wedge \neg P_6 \wedge P_7 \wedge \neg P_8 \tag{2.3}$$

The transition relation is a disjunction of the different changes that are possible. In a Petri net, each such a possible change is due to a Petri net transition, and both the pre- and post-conditions are defined by the arcs that connect to it. For t_1 the proposition is given below.

$$\begin{aligned} enc(t_1) := P_1 \wedge \neg P_1' \wedge \neg P_2 \wedge P_2' \wedge P_3 \Leftrightarrow P_3' \wedge P_4 \Leftrightarrow P_4' \\ \wedge P_5 \Leftrightarrow P_5' \wedge P_6 \Leftrightarrow P_6' \wedge P_7 \Leftrightarrow P_7' \wedge P_8 \Leftrightarrow P_8' \end{aligned} \tag{2.4}$$

The proposition for the transitions as a whole is $\bigvee_{t \in T} enc(t)$ which combines each of these steps into one set of mutually exclusive possibilities. In order to define the fact that no transition can be made that leaves the goat on one shore with the wolf or cabbage without the ferryman, T is conjoined with the following rule.

$$\neg(P_5' \wedge \neg P_1' \wedge (P_3' \vee P_7')) \vee \neg(P_6' \wedge \neg P_2' \wedge (P_4' \vee P_8')) \tag{2.5}$$

The property (ϕ) for the method to check is the absence of the win condition. This way, if the T was encoded correctly, the PDR implementation will return a path towards the win condition. Since the win condition is each entity being on the right shore, the property the program is asked to check is the opposite of that.

$$enc(\phi) := \neg(\neg P_1 \wedge P_2 \wedge \neg P_3 \wedge P_4 \wedge \neg P_5 \wedge P_6 \wedge \neg P_7 \wedge P_8) \tag{2.6}$$

This example does return the right answer to the riddle. It can be found at https://github.com/OlafRandel/Z3_petrinet as the class "Test.java". From this template, the practice of forming extensive conjunctions in the encoding of Petri nets is derived.

3 Encoding

For the application of a PDR method for a Petri net (P, T, A), three boolean expressions need to be generated. These Boolean expressions stand for the initial position, transition relation, and relevant property respectively.

3.1 Initial Position

The initial position is a list of atomic Boolean values, corresponding to the initial marking (m_0) of the Petri net. There is only one satisfying valuation of I. In the equation below every individual place is given the same value in I as it has in m_0

$$\bigwedge_{p \in P} p \Leftrightarrow m_0(p) \tag{3.1}$$

3.2 Transition Relation

The transition relation (TR) is a disjunction of every transition in the Petri net. For every place in the Petri net, p, we reserve two Boolean variables, (p) and (p′). In this notation if (a) is true, that means that before the transition place a contained a token. If (a′) is true, the place contains a token after the transition has been fired. If place a is not affected by the transition, that would be denoted as $(a) \Leftrightarrow (a')$. In summary, our encoding of the transition relation for the net is a Boolean formula TR, such that all satisfying assignments represent the edges in the reachability graph (m_0, M, E).

$$TR(p_1, .., p_n, p'_1, .., p'_n) \iff \langle\langle p_1, .., p_n, \rangle, \langle p_1, .., p_n, \rangle\rangle \in E \tag{3.2}$$

To generate such an encoding, we iterate over target and source places of individual transitions $t \in T$. To do so, we use the notation $\circ t$ and $t\circ$. $\circ t := \{p \in P \mid (t, p) \in A\}$ and $t\circ := \{p \in P \mid (p, t) \in A\}$. We will now explain the three main conjuncts for every transition. If a place is a source place ($p \in \circ T$) the transition can fire when it contains a token and causes it to become empty. If a place is a target ($p \in t\circ$), the transition can fire when the place is empty and causes it to be gain a token. And for every place in the net that is unrelated to this transition ($p \notin (\circ T \cup t\circ)$, the variable must remain unchanged. In equation 3.3 we give the encoding of a single transition, here we take every place p of the Petri net, and set the value of the corresponding Boolean variables (p) and (p′).

$$enc(t) := (\bigwedge_{p \in \circ t} p \wedge \neg p') \wedge (\bigwedge_{p \in t\circ} \neg p \wedge p') \wedge (\bigwedge_{p \notin (\circ t \cup t\circ)} p \Leftrightarrow p') \tag{3.3}$$

However, this definition is imperfect. In Petri nets a place can be connected to both an outgoing arc and an incoming arc of the same transition. If this is the case, the place does not need to be empty as a destination beforehand, or empty as a source after the fact. The places that appear in both stay full. Equation 3.4 shows the amended and complete symbolic representation of a single transition.

$$enc(t) := ((\bigwedge_{p \in \circ t \setminus t\circ} p \wedge \neg p') \wedge (\bigwedge_{p \in t\circ \setminus \circ t} \neg p \wedge p') \wedge (\bigwedge_{p \notin (\circ t \cup t\circ)} p \Leftrightarrow p') \wedge (\bigwedge_{p \in \circ t \cap t\circ} p \wedge p')) \tag{3.4}$$

The transition relation TR is a disjunction of all these transitions. We can therefore combine all these sub-definitions into one large predicate. In equation 3.5 the previous equation is given for every transition in the Petri net.

$$TR := \bigvee_{t \in T} enc(t) \tag{3.5}$$

3.3 Property

The property that must be proven invariant (ϕ) stands for what the model is being tested for. In the case of Petri nets there are various properties that a user might want to verify. For the sake of this research we have encoded one. The 1-safety.

3.3.1 1-safety

A Petri net is 1-safe if a place can not contain more than one token. The property we define is therefore the quality of no transition being fireable while one of its targets already contains a token. In other words, the property means that it is not true that for any of the transitions that all the sources are true, and one or more of the targets are true. We encode this requirement for each transition as $enc_{safe}(t)$ in equation 3.6.

$$enc_{safe}(t) := \neg((\bigwedge_{p \in \circ t} p) \wedge (\bigvee_{p \in t \circ} p)) \tag{3.6}$$

Again, we must account for places that are both sources and targets. These may be true while keeping the net 1-safe. The amended encoding for each transition is shown in equation 3.7.

$$enc_{safe}(t) := \neg((\bigwedge_{p \in \circ t} p) \wedge (\bigvee_{p \in t \circ \setminus \circ t} p)) \tag{3.7}$$

The 1-safety property is true for the net as a whole if this encoding is true for all transitions simultaneously. We can therefore encode it as a conjunction over all transitions.

$$enc(\phi) := \bigwedge_{t \in T} enc_{safe}(t) \tag{3.8}$$

4 Experimentation

We have implemented the encoding of Petri nets as described in chapter 3 using the java programming language. It can be found at `https://github.com/OlafRandel/Z3_petrinet` as the class 'interpretCheck.java'. This implementation interprets pnml files to run on the Z3 implementation of PDR [4].
At first, we had planned to utilize pnml problems from the annual Model Checking Competition for this experiment. However, it soon became apparent that the method would not be able to complete any of these in a reasonable time. Smaller custom tests were made, pre-encoded for the PDR method and in the .lola language for a competing model checking tool. LoLa is an existing model checking tool that used for Petri nets [5]. Figure 4.1 should show the strong disparity between the two methods. The difference in calculation time increases exponentially with complexity.

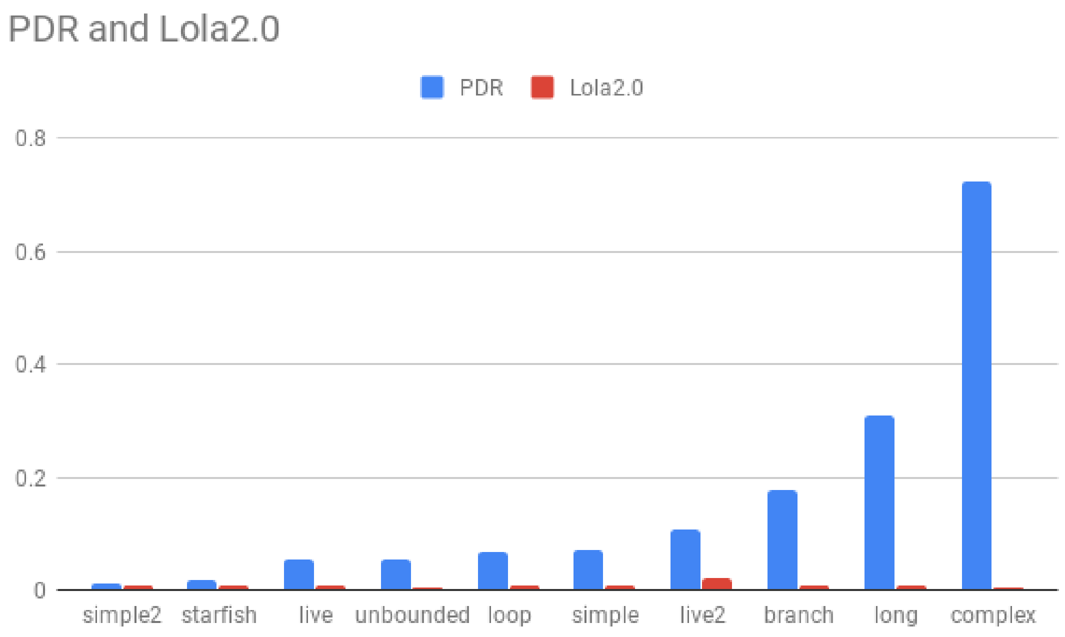

Figure 4.1: Results of the initial experiment.

The tests for lola and PDR can be found here `https://github.com/OlafRandel/lola2.0_customTests`, they are direct translations of the 'customTests.java' unit tests in `https://github.com/OlafRandel/Z3_petrinet`.

5 Conclusion

The answer to the research question is: We can prove the correctness of a Petri nets using property directed reachability by interpreting the initial state, the transition relation, and the provable property into symbolic logic, using the distinct transitions as a guide.
The initial experimentation has shown a negative result on the efficiency of this method. Although not conclusive it suggests the method is inefficient for these problems.
In Chapter 1 we described how, because of the locality of the changes Petri nets undergo when a transition fires, they seemed a good subject to use PDR on. However, it could be that since the firing of any transition in a Petri net can be followed by the firing of any other transition at any place in the net the changes it undergoes are not so local over time.
Possible follow-up research could more closely examine the performance of this method with different kinds of Petri nets.

Bibliography

[1] Baier, C., Katoen, J. and Larsen, K. (2008) *Principles of Model Checking*. MIT press.

[2] Rozenburg, G.; Engelfriet, J. (1996). *Elementary Net Systems* In: Reisig W., Rozenberg G. (eds) *Lectures on Petri Nets I: Basic Models.* ACPN 1996. Lecture Notes in Computer Science, vol 1491. Springer, Berlin, Heidelberg.

[3] Günther, H., Laarman, A., Weissenbacher, G. (2016, April). Vienna verification tool: IC3 for parallel software. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. (pp. 954-957). Springer, Berlin, Heidelberg.

[4] Nikolaj Bjorner, Leonardo de Moura (2008) *Z3: An Efficient SMT Solver* In: Ramakrishnan C.R., Rehof J. (eds) *Tools and Algorithms for the Construction and Analysis of Systems.* TACAS 2008. Lecture Notes in Computer Science, vol 4963. Springer, Berlin, Heidelberg.

[5] Schmidt, K. (2000, June). Lola a low level analyser. In *International Conference on Application and Theory of Petri Nets* (pp. 465-474). Springer, Berlin, Heidelberg.