Symbolic, Symmetry, and Stubborn Set Searches

Mikko Tiusanen
Helsinki University of Technology
Otakaari 1A, SF–02150 Espoo, FINLAND

November 11, 1993

Abstract

The *state space explosion problem* is the proliferation of states to be considered during the verification of a finite state system. This paper proposes ways to combine three methods that have successfully been used to alleviate this problem during the reachability analysis of safe Petri nets (or ones with known bounds for all places): the *symbolic model checking* employing data structures for *binary-decision diagrams* (BDDs), the symmetry equivalence method for Petri nets of Jensen et al, and the *stubborn set method* of Valmari.

The use of *Petri nets* as the basis extends the scope of the BDD-based analysis beyond hardware verification. The reachability graph representation of the state space of Petri nets is slightly generalized to allow for multiple initial markings, in order to correspond to the symbolic state space search using BDDs. The generalization is useful in itself if the initial marking of the system is not fully defined, e.g. for fault-tolerant systems. A stubborn set selection algorithm of Valmari is generalized to apply to sets of current markings instead of one marking at a time. This will provide the combination of the stubborn set method and symbolic state space generation using BDDs. The combination of the symmetry equivalence method with the stubborn set method is discussed in more detail than previously. Using BDDs to benefit in the symmetry equivalence method is also discussed.

Topic: Analysis of the behaviour of nets (finite state verification).

1 Introduction

The *state space explosion problem* refers to the effect of having to inspect a large—computationally infeasible—number of states during the verification of a finite state system, be it hardware, software, parallel, concurrent, distributed, real-time, or some combination of these.

We shall propose a way to combine two successful methods to alleviate the state space explosion problem in *reachability analysis*, that is, explicit generation of the state space of the system, namely, the *stubborn set method* by Valmari [21, 22, 20], and the *symbolic model checking* methods employing graph data structures for *binary-decision diagrams* (BDDs) [6, 7, 4, 5]. The BDDs have been successfully used to verify hardware systems with more than 10^{20} states. The stubborn sets, on the other hand, offer in some systems a reduction from a number of states exponential in a number of independent agents to polynomial, even linear, by reducing the number of interleavings of actions or events that need to be considered. The method has so far been restricted to generating the state space one global state at a time.

One of the first methods of doing high-level analysis of high-level Petri nets was the so called *equivalent marking* or *symmetry equivalence method* proposed in [11] for Coloured Petri nets. It uses *symmetries* to reduce the number of markings considered during the reachability graph generation. We shall follow the terminology of [19] rather than that of [11]. We will discuss in detail how to combine this with the stubborn set method. We will also discuss some ways of improving the symmetry method through the use of BDDs.

We shall use *safe Petri nets* [16, 2, 3] (or more generally Petri nets with known bounds on all places) as the system model that the methods are applied on. The use of Petri nets extends the applicability of BDDs to non-hardware systems. The combination with stubborn sets potentially reduces the number of redundant interleavings considered, which has still been a problem with applications of BDDs to asynchronous hardware [6]. We shall define a simple generalization of the reachability graph of Petri nets to multiple initial markings that is potentially useful for fault-tolerant systems as such. We shall use this definition as the basis for discussing the symbolic model checking through reachability analysis.

In the sequel we shall first review some of the relevant notation and definitions. We shall derive a symbolic search algorithm using Dijkstra's *weakest precondition* or $\wp$*-calculus*, and discuss the combinations of the methods before conclusions and considering future work.

2 Notations and definitions

In this section we shall present some notation on Petri nets used in the sequel when presenting the algorithms. We will also briefly review the relevant points of binary-decision diagrams and stubborn sets. We shall use Dijkstra's *language of guarded commands* and his *weakest precondition* or $\wp$*-calculus*, see e.g. [10], in presenting the algorithms. We will use the notation $(\forall x : p(x) : q(x))$ to denote the universal quantification, more commonly given as $\forall x : (p(x)) \Rightarrow (q(x))$. Similarly, $(\exists x : p(x) : q(x))$ denotes the existential quantification $\exists x : (p(x)) \wedge (q(x))$. In these, the predicate $p(x)$ above is called the *quantification range*. The notation is motivated by the benefts for the intuition resulting from having the range as a separate entity. Furthermore, the application of De Morgan's laws is somewhat simpler since the range need not be changed.

The proofs we will present will be calculations of the form

$$\begin{array}{ll} & A \\ = & \{\text{hint as to why } A \Leftrightarrow B\} \\ & B \\ = & \{\text{hint as to why } B \Leftrightarrow C\} \\ & C, \end{array}$$

the conclusion being reached being that $A \Leftrightarrow C$, since $(A \Leftrightarrow B) \wedge (B \Leftrightarrow C)$. Note that this proof cannot be used to conclude $A \Leftrightarrow B \Leftrightarrow C$. Some or all of the equalities can be replaced with implications uniformly in one direction to conclude an implication in the same direction between the top and bottom predicates. This relies on the tautology $((A \Rightarrow B) \wedge (B \Rightarrow C)) \Rightarrow (A \Rightarrow C)$. We shall assume of the Boolean operators that negation $\neg$ has the highest priority, then conjunction $\wedge$ and disjunction $\vee$ at the same priority, then implication $\Rightarrow$ (or $\Leftarrow$), and finally equivalence $\Leftrightarrow$. Parentheses must be used to make any expression containing both $\wedge$ and $\vee$, or multiple occurrences of implication unambiguous. Keyword "splitting" refers to doing a case analysis on the elements of a set or the range of a quantification.

We shall use the notation R^* to denote the reflexive and transitive closure of a binary relation $R \subseteq X^2$ over the set X. We will also use it as the Kleene star, denoting the set of finite strings over a set, depending on the context.

2.1 Petri nets

There are many closely related formal definitions of Petri nets. We shall use the following, see e.g. [15]. Let N be the set of natural numbers, with 0 as the least element.

Definition 1 A *Petri net* N is a 5-tuple $N = \langle S, T; F, W, m_0 \rangle$ where

- S is a finite set of *places*,
- T is a finite set of *transitions*,
- $F \subseteq (S \times T) \cup (T \times S)$ is a set of arcs, the *flow* relation,
- $W : F \to (\mathsf{N} \setminus \{0\})$ is the *weight function* on the arcs, and
- $m_0 : S \to \mathsf{N}$ is the *initial marking*, giving the initial distribution of tokens to places,

such that the sets of places and transitions are disjoint, $S \cap T = \emptyset$, and there is at least one net element, $S \cup T \neq \emptyset$. The domain of the weight function W is usually extended to $(S \times T) \cup (T \times S)$ by setting $W(\langle x, x' \rangle) = 0$ iff $\langle x, x' \rangle \notin F$. Usually, the value of either of these arc weight functions on $\langle x, x' \rangle$ is denoted simply as $W(x, x')$. Any function $m : S \to \mathsf{N}$, giving the distribution of the tokens onto places is called a *marking* of the net in question. The set of markings of a net with the places S is denoted by M_S. For any net element $x \in S \cup T$, the *preset* (input elements) of x is ${}^\bullet x = \{x' \in S \cup T \mid x' \; F \; x\}$ and the *postset* (output elements) of x is $x^\bullet = \{x' \in S \cup T \mid x \; F \; x'\}$. A net is called *simple* iff there are no two net elements that have identical presets and identical postsets, respectively. If there is a pair of net elements x, x' such that $x \in {}^\bullet(x')$ and $x' \in {}^\bullet x$, $\boxed{} \leftrightarrows \bigcirc$, such a pair is called a *self-loop*. A Petri net that contains no self-loops in called *pure*. $\diamond$

The definition of a post- or a preset of an element can obviously be extended to apply to a set of elements by taking the union of the post- or presets of the elements of the set, respectively. The behavior of a Petri net is defined as follows, see e.g. [15]:

Definition 2 A transition t of a Petri net $N = \langle S, T; F, W, m_0 \rangle$ is *enabled at a marking* $m : S \to \mathsf{N}$ iff there are enough tokens on the input places:

$$\forall s : s \in {}^\bullet t \Rightarrow m(s) \geq W(s, t).$$

This is denoted by $m\ [t\rangle$. An enabled transition $t \in T$ may, but need not *occur* (fire); if it does, it transforms the marking m to another m' that satisfies

$$\forall s : s \in S \Rightarrow m'(s) = m(s) + W(t, s) - W(s, t).$$

The relation that m' is obtained from m by firing transition t is denoted by $m\ [t\rangle\ m'$; this implies $m\ [t\rangle$. We shall use $m\ [\rangle\ m'$ to denote that there is a transition t such that $m\ [t\rangle\ m'$. ◇

In effect, the firing of a transition removes tokens from the input and deposits tokens on the output places of the transition, as many as the arc weights require. For a simple net, if there is a transition t such that $m\ [t\rangle\ m'$, it is *unique*.

Definition 3 The *reachability graph* of a Petri net $\langle S, T; F, W, m_0 \rangle$ is the (labeled) flowgraph $\langle V, A; r \rangle$, where

- $V = \{m \in \mathsf{M}_S \mid m_0\ [\rangle^*\ m\}$ is the set of markings (finitely) reachable from the initial marking,
- $A = \{\langle m, t, m' \rangle \mid m \in V \wedge m\ [t\rangle\ m'\}$ are the arcs labelled with the transition involved, and
- $r = m_0$.

A Petri net is called *k-bounded* iff there is at most k tokens in any place of the net in any reachable marking. A 1-bounded net is called *safe*. ◇

We will generalize the concept of a reachability graph to better suit our purposes. This generalized reachability graph is potentially useful when discussing systems that do not have a fully defined initial (or representative) state. Fault-tolerant systems that contain a model of the faulty conponents (an explicit fault model) can be such. The usefulness stems from the fact that the generalized reachability graph is the union of the vertex and arc sets of all the reachability graphs of any marking in a given finite set of initial markings. Instead of doing separate reachability analyses for each of the initial markings, parts of the graphs can be shared during the analysis.

Assume that a finite set M of initial markings for the net N is given.

Definition 4 The *generalized reachability graph* of a Petri net with a set of initial markings M is the directed graph $\langle V, A \rangle$ rooted at M, where

$$V = \{m \in \mathsf{M}_S \mid (\exists m_0 : m_0 \in M : m_0\ [\rangle^*\ m)\}$$

is the set of markings (finitely) reachable from an initial marking, and

$$A = \{\langle m, t, m' \rangle \mid m \in V \wedge m\ [t\rangle\ m'\}$$

are the arcs labelled with the transition. ◇

Obviously, this is a very slight generalization: by adding one place and $|M|$ transitions, assuming this is finite, one can from any finite Petri net construct another with exactly one extra state, namely the initial one, and excluding that marking, an isomorphic (generalized) reachability graph (whether finite or not).

The language of the net N, that is, the set of (finite) firing sequences that can be fired from an initial marking, is denoted by

$$L(N, M) = \{\sigma \in \textstyle\sum \mid (\exists m_0 : m_0 \in M : m_0\ [\sigma\rangle\ )\}.$$

Note that the first marking of any $\sigma \in L(N, M)$ is from the set M. Also note that the empty sequence ϵ is not a transition sequence.

2.2 Stubborn sets

We shall describe what is called by Valmari [22] the *strong* theory of stubborn sets. It aims at using knowledge about the concurrency, or, more generally, independence or non-interference of transitions, to reduce the number of elements of $X^\bullet$ actually considered further during the symbolic search, say the cardinality of the set Y. This will bring down the *branching factor* of the search even if it does not minimize the number of markings generated. What is characteristic of the behavior of concurrently enabled transitions apparent in a reachability graph of a Petri net is the *diamond substructure*: if transition t and t' are concurrently enabled at a marking m they will span a square

$$\begin{array}{ccc} m & \stackrel{t}{\rightarrow} & m' \\ t' \downarrow & & \downarrow t' \\ m'' & \stackrel{t}{\rightarrow} & m''' \end{array}$$

in the reachability graph. Any set of $n > 0$ concurrently enabled transitions at a marking will span an n-dimensional hypercube. But not only do concurrently enabled transitions span these squares: some seemingly conflicting transitions will span these, too. These squares are, in fact, commutative diagrams. The important thing to note is that these transitions form a "well-behaved" part of the nondeterminism included in the interleaved interpretation of concurrency, which the reachability graph is based on. There is little to be done but to explore all the choices of a "truly" nondeterministic decision such as a conflict in a Petri net or other resource allocation to competitors. On the other hand, concurrently enabled or—more generally—commuting transitions offer a possibility of avoiding the generation of at least one of the four markings in a square: final state of the square is guaranteed to be the same whichever way is taken to reach it. In a hypercube of higher dimension, say n, any path through it can be chosen to dissect it, resulting in a number of markings linear in n to be generated, instead of a number exponential in n. Naturally, there might be states of interest within the hypercube, say, because they enable yet another transition in exactly that state. Since such a transition might be on the only path to a deadlock, it must be considered and the state in the middle must be generated while generating the reachability graph. The challenge is to formulate concepts that allow making use of commutativity among transitions to reduce the size of Y while preserving the completeness of the analysis in the sense of finding all the deadlocks.

Consider a marking, say m, whether reachable or not, that has at least one enabled transition. Now, try to construct a partition of the the set of transitions T into two sets, say T_s and $\overline{T_s}$, so that the following diagram commutes for any $t \in T_s$ and $\sigma \in (\overline{T_s})^*$ such that $m\ [t\rangle\ \wedge m\ [\sigma\rangle$:

$$\begin{array}{ccc} m & \stackrel{\sigma}{\rightarrow} & m' \\ t \downarrow & & \downarrow t \\ m'' & \stackrel{\sigma}{\rightarrow} & m''' \end{array}$$

If one furthermore can guarantee that firing any $\sigma \in (\overline{T_s})^*$ cannot enable any transition in T_s, there is no need to generate any of the internal markings on the path $m\ [\sigma\rangle\ m'\ [t\rangle\ m''$: any deadlock that can be found by generating such a path, can be found by firing some enabled transition $t \in T_s$, if such exists. If one of the blocks contains no enabled transitions, we have gained nothing. But if both contain enabled transitions, we can ignore any enabled transition in $\overline{T_s}$ when choosing Y, since firing any of them will not affect the enabledness of any of the stubborn transitions (the stubborn transitions will stay enabled/disabled); firing a stubborn transition will at most enable new transitions. The only thing that remains is to ensure that there is at least one transition in T_s enabled at marking m to make T_s a *stubborn* set. Valmari has given many alternative definitions for stubborn sets of Petri nets. We shall use the following [22]:

$$\begin{aligned} \mathit{stubborn}(T_s, m) = T_s \subseteq T \wedge (\exists t : t \in T_s : m\ [t\rangle\ ) \\ \wedge (\forall t : t \in T_s : [m\ [t\rangle\ \Rightarrow ({}^\bullet t)^\bullet \subseteq T_s] \\ \wedge [\neg m\ [t\rangle\ \Rightarrow (\exists s : m(s) < W(s,t) : {}^\bullet s \subseteq T_s)]) \end{aligned}$$

Obviously, T is stubborn at any marking with at least one enabled transition. Note that this definition mixes structural elements, e.g. $({}^\bullet t)^\bullet \subseteq T_s$, with elements depending on the marking, e.g. $m(s) < W(s,t)$. So the goal is the set of enabled transitions of a stubborn set:

$$R : \mathit{stubborn}(T_s, m) \wedge Z = T_s \cap \{t \in T \mid m\ [t\rangle\ \}.$$

The definition is suggestive of an algorithm to compute a stubborn set at a marking: given a method to pick a place s in the definition, the *scapegoat*, any marking m determines a binary relation $t\ R_m\ t'$: "if t is in a stubborn set, so is

t', immediately due to definition". Starting from an enabled transition t and computing the set of enabled transitions in the first-found fragment of the graph (T, R_m) that really has an enabled transition using Tarjan's algorithm, we have the set of enabled transitions of a stubborn set. The algorithm can be stopped after having found the first fragment.

The correctness of the algorithm relies on the fact that Tarjan's fragment search algorithm will find a fragment only after all the fragments that can be reached from it have been found. If the algorithm is stopped as soon as it finds a fragment with at least one enabled transition, the fragment *together with* all the fragments that can be reached from it forms a stubborn set. However, we are only interested in the enabled transitions in the stubborn set, and the other fragments reachable from the found one do *not* contain enabled transitions since these were found earlier, so the others can be ignored. One of the interesting aspects of this algorithm is that it inspects at most once the enabledness of any transition, making it at most linear in the number of transitions. We give an iterative version of this algorithm in Appendix A.

In the sequel, we shall use $stub(m)$ to denote a stubborn set computed by the given algorithm at the marking m, $m\ [t\rangle_s$ to denote $m\ [t\rangle\ \wedge t \in stub(m)$, $m\ [t\rangle_s\ m'$ to denote $m\ [t\rangle\ m' \wedge t \in stub(m)$, and $m\ [\rangle_s\ m'$ to denote $(\exists t : t \in stub(m) : m\ [t\rangle\ m')$. The notation $m\ [\sigma\rangle_s\ m'$ is used to denote that marking m' can be reached from marking m with a sequence of transitions σ that has each of its transitions in the chosen stubborn set of the marking it is fired from in the sequence (a *stubborn transition sequence*), and the notation $m\ [A\rangle_s{}^*\ m'$ is used to denote that marking m' can be reached from marking m by a stubborn transition sequence through markings in the set A only, excluding the first marking m, however. Finally, A° is used to denote the stubborn postset of the set A, $\{y \mid (\exists x, t : x \in A \wedge t \in stubborn(x) : x\ [t\rangle\ y)\}$.

2.3 Binary-decision diagrams

Binary-Decision Diagrams (BDDs) [12, 1] are full binary trees that represent the result of a sequence of two-way (binary) choices. Bryant [4] introduced a graph representation for these that can be used to represent Boolean functions of Boolean variables. The representation imposes (or assumes) a linear order on the argument variables; the resulting representation is unique, providing a *canonical* representation of functions. Given any canonical representation of Boolean functions of Boolean variables, this can also be seen to be a representation of sets of binary vectors, a set being represented by the *characteristic function* of the set. The characteristic function is a function that has the value **true** on those binary vectors that are in the set under consideration, and **false** on all other vectors of same dimension.

The representation proposed by Bryant consists of imposing a total order on the Boolean variables appearing as arguments to a function and then systematically applying two transformations to any BDD. The order on variables determines the order of the decisions made in the BDD, starting from the root. First of the transformations makes the vertices of the resulting graph *share* any subgraph representing identical subtrees of the original full binary graph. That is, all of the identical subtrees of the original BDD are represented by a unique subgraph of the resulting graph, but the subgraph may be the immediate descendant of many vertices. Since the order on variables (decisions) is total, this results in a rooted directed, acyclic graph (DAG) where all the vertices at the same level (distance from the root) represent a choice made based on the same variable. The second transformation replaces any vertex, both the descendants of which represent the same subtree, by the representation of the common subtree. The replaced vertices are redundant, since the choice made does not affect the value of the function represented: the value is independent of the variable. Since after this transformation the distance to the root is no longer sufficient to determine the variable based on which the choice is made, this must somehow be attached to a vertex explicitly. These two transformations are conceptually repeated until they can no longer be applied.

The interest in the representation stems from the fact that it can be used to efficiently perform operations on a sets of binary vectors, such as testing for emptiness, union, intersection, and set difference. The functional composition of functions can also be performed as an operation on the representations of these functions. Since the state of a digital circuit can on a particular level of abstraction be represented as a binary vector, the data structure has been used with success for hardware verification [5, 7, 6]. One of the keys for this success has been the realization that the next-state relation of a digital circuit can also be represented using the data structure, and that the application of this relation to a set of current states can be used to produce the set of successor states of any current state. This provides a way of doing a set-parallel, breadth-first search of the state space of a digital circuit. The efficiency of the search depends heavily on the choice of the ordering of variables, however.

3 Symbolic search

Consider performing a search of the generalized reachability graph of a Petri net. We shall start by applying the well-known white-grey-black coloring invariant of Dijkstra's. Let Q be the predicate stating that V is the set of vertices of the generalized reachability graph of the net N:

$$Q : V = \{m \in \mathsf{M} \mid (\exists m_0 : m_0 \in M : m_0\ [\rangle^*\ m)\}.$$

We shall not consider the computation of the arcs of the generalized reachability graph: this is a straightforward extension of the resulting algorithm. The objective is then to compute the sets of vertices of the graph to a variable, say B: $R : B = V$. To get the graph we must somehow generate the vertices, which calls for a loop. Following Dijkstra, we shall choose the invariant of the loop to consist of two parts. First, the (still to be determined) vertices of the graph are partitioned into three blocks: the black, the grey and the white vertices, with the result being computed as the set of black vertices,

$$P_0 : V = B \uplus G \uplus W.$$

Recall that $\uplus$ is used to denote the union of two disjoint sets. The intuition is to have the black ones be the processed, the grey the still possibly unprocessed, and the white the unknown vertices of V: the algorithm is to "color" all the vertices black. Note that $G = W = \emptyset \wedge P_0 \Rightarrow R$, so we have not only weakened the postcondition to obtain the invariant, but also found a (tentative) termination condition $G = W = \emptyset$. Initialization for the loop is obtained easily, as

$$\begin{aligned}
&\wp(\text{"}B, G, W := \emptyset, M, V \setminus M\text{"}, P_0)\\
&= V = \emptyset \uplus M \uplus (V \setminus M)\\
&= V = M \uplus (V \setminus M)\\
&= \{\text{set theory}\}\\
&\quad M \subseteq V\\
&\Leftarrow Q.
\end{aligned}$$

Now, it is hard to check the given termination condition, especially since the set W is not known. Again following Dijkstra, we shall proceed with the idea that the grey vertices shall be a "buffer zone" between the other two blocks: $\boxed{W \boxed{G \boxed{B}}}$. To formulate this, we shall require that for any white (unknown) vertex there be a white path from some grey (unprocessed) vertex to it.

$$P_1 : (\forall w : w \in W : (\exists g : g \in G : g\ [W\rangle^*\ w)).$$

Note that now

$$\begin{aligned}
&P_1 \wedge G = \emptyset\\
&\Rightarrow (\forall w : w \in W : (\exists g : g \in \emptyset : g\ [W\rangle^*\ w))\\
&= (\forall w : w \in W : \mathbf{false})\\
&= W = \emptyset.
\end{aligned}$$

Using the $P_0 \wedge P_1$ as the invariant, we get

$$\begin{aligned}
&P_0 \wedge P_1 \wedge G = \emptyset\\
&\Rightarrow \{\text{see above}\}\\
&\quad P_0 \wedge W = G = \emptyset\\
&\Rightarrow R.
\end{aligned}$$

We then must check the initialization of P_1:

$$\begin{aligned}
&\wp(\text{"}B, G, W := \emptyset, M, V \setminus M\text{"}, P_1)\\
&= (\forall w : w \in V \setminus M : (\exists g : g \in M : g\ [V \setminus M\rangle^*\ w))\\
&\Leftarrow \{(*)\}\\
&\quad (\forall w : w \in V : (\exists g : g \in M : g\ [V\rangle^*\ w))\\
&= \{\text{change of variable, } [V\rangle = [\rangle\}\\
&\quad (\forall w : w \in V : (\exists m_0 : m_0 \in M : m_0\ [\rangle^*\ w))\\
&\Leftarrow V = \{m \in \mathsf{M} \mid (\exists m_0 : m_0 \in M : m_0\ [\rangle^*\ m)\}\\
&\Leftarrow Q.
\end{aligned}$$

Here (*) holds because repeating a vertex, $g \in M$ in this case, does not help if there is no path between two particular vertices, as the choices for arcs do not *increase* beyond those on the first encounter of a vertex no matter how many times the vertex is entered. This is trivial for an explicit graph, but it is an important property of an

implicitly defined graph. In a sense, the notion of a vertex for an implicitly defined graph is not well defined, if the number of arcs out of it can increase beyond that on the first encounter based on how many times it is entered. As far as computing the set of reachable vertices is concerned, it is irrelevant whether the number of choices of arcs ever *decreases* or not, as long as it never exceeds that on the first encounter: all the reachable vertices will be encountered in any case.

As the termination function for the loop, $|V \setminus B|$ is an obvious choice: $|V|$ should, therefore, be finite for the loop to terminate, and the body of the loop should color black at least one vertex that was not black previously. Based on the intuition about P_1, consider then coloring some non-empty subset of vertices $\emptyset \neq X \subseteq G$ black: this might result in some of the white neighbours of X, $X^\bullet \cap W$, not to have a grey vertex from which these can be reached. To avoid this, color all the white neighbours of X grey. (Note in the following that $W \setminus (X^\bullet \cap W) = W \setminus X^\bullet$.)

$$
\begin{array}{ll}
 & \wp(\text{“}B, G, W := B \cup X, (G \setminus X) \cup (X^\bullet \cap W), W \setminus X^\bullet\text{”}, P_0 \wedge P_1) \\
= & V = (B \cup X) \uplus ((G \setminus X) \cup (X^\bullet \cap W)) \uplus (W \setminus X^\bullet) \\
 & \quad \wedge (\forall w : w \in W \setminus X^\bullet : (\exists g : g \in (G \setminus X) \cup (X^\bullet \cap W) : g\ [W \setminus X^\bullet\rangle^*\ w)) \\
\Leftarrow & P_0 \wedge \emptyset \neq X \subseteq G \\
 & \quad \wedge (\forall w : w \in W \setminus X^\bullet : (\exists g : g \in (G \setminus X) \cup (X^\bullet \cap W) : g\ [W \setminus X^\bullet\rangle^*\ w)) \\
\Leftarrow & \{(**)\} \\
 & \quad P_0 \wedge P_1 \wedge \emptyset \neq X \subseteq G
\end{array}
$$

where (**) is the result of the fact that any white path from X to $y \in W$ will pass through some vertex $z \in X^\bullet \cap W$, and any path passing through some vertex $z \in X^\bullet \cap W$ needed satisfy $g\ [W\rangle^*\ y$ for some other $g \in G$, will imply a path $z\ [W \setminus X^\bullet\rangle^*\ y$, which maintains P_1, since $X^\bullet \cap W \subseteq (G \setminus X) \cup (X^\bullet \cap W)$. Therefore, $P_1 \wedge \emptyset \neq X \subseteq G$ implies the last conjunct.

Setting $P : P_0 \wedge P_1$, assuming there is a function *pick* satisfying

$$G' = G \neq \emptyset \Rightarrow \wp(X := pick(G), \emptyset \neq X \subseteq G = G')$$

for any G', and collecting the results of the analysis, we have

$$
\begin{array}{ll}
 & \{Q\}\ \ B, G, W, E := \emptyset, M, V \setminus M, \emptyset \{P\} \\
; & \textbf{do}\ \ G \neq \emptyset \rightarrow \\
 & \quad X := pick(G) \{P \wedge \emptyset \neq X \subseteq G\} \\
 & ; B, G, W, E := B \cup X, (G \setminus X) \cup (X^\bullet \cap W), \\
 & \quad W \setminus X^\bullet, E \cup \{\langle m, t, m' \rangle \mid m \in X \wedge t \in T \wedge m\ [t\rangle\ m'\} \\
 & \ \textbf{od} \\
 & \{P \wedge G = \emptyset\}\ \ \{R\}
\end{array}
$$

Note that this program subsumes many search strategies, since it does not restrict the implementation of *pick* to any great extent. Most common implementations of the algorithm make M contain a single element and *pick* return a single element. Moreover, any implementation for the necessary set operations is equally valid, but will naturally have an effect on the performance of the search. In particular, the data structures for binary-decision diagrams (BDDs) can be used to implement the sets of markings B and G for safe Petri nets. Of course, since V is not known prior to the search, the set W will have to be implemented implicitly as the set of reachable markings not already colored, that is, not in B or G. One of the powerful features of BDDs is that the transition relation among global states can be represented as a BDD, and, furthermore, to compute the *set* of followers of a given *set* of current global states. For Petri nets, this involves having a BDD representation of the set of binary vectors of the relation

$$\{\langle m, t, m' \rangle \mid m, m' \in \mathsf{M} \wedge t \in T \wedge m\ [t\rangle\ m'\}.$$

This can be used to compute the followers of a given set of markings also represented as a BDD by using operations on the two BDDs. For a Petri net with known bounds on the marking of any place, the marking of the places can be suitably encoded as binary variables to achieve the same result. In effect, we shall assume the use of the graph representation of BDDs as an efficient implementation of operations on sets of markings and the operation $X^\bullet$ for a set X of markings. We shall return to some of the aspects of the use of BDDs in Section 6.

Note that the program also maintains the invariant

$$P_3 : B^\bullet \subseteq B \cup G,$$

since

$$\wp(\text{“}B, G, W := \emptyset, M, V \setminus M\text{”}, P_3)$$

$= \emptyset^\bullet \subseteq \emptyset \cup M$
$= \emptyset \subseteq M$
$=$ **true**
$\Leftarrow Q$,
and
$\wp(\text{“}B, G, W := B \cup X, (G \setminus X) \cup (X^\bullet \cap W), W \setminus X^\bullet\text{”}, P_3)$
$= (B \cup X)^\bullet \subseteq B \cup X \cup (G \setminus X) \cup (X^\bullet \cap W)$
$\Leftarrow X \subseteq G \wedge (B \cup X)^\bullet \subseteq B \cup G \cup (X^\bullet \cap W)$
$\Leftarrow X \subseteq G \wedge P_3 \wedge X^\bullet \subseteq B \cup G \cup (X^\bullet \cap W)$
$\Leftarrow$ {case analysis on the color of $y \in X^\bullet$}
$X \subseteq G \wedge P_0 \wedge P_3$.

4 Symbolic stubborn set search

We shall present an algorithm extending the stubborn set method for the computation of the generalized reachability graph of safe Petri nets. The algorithm given in Section 3 will be used as a basis. As can easily be checked, this provides also a program to do the stubborn set search, after replacing $[\ldots\rangle$ by $[\ldots\rangle_s$ and $(\ldots)^\bullet$ by $(\ldots)^\circ$ in the program and in the derivation. Intuitively, the stubborn set search merely generates the subgraph of the full reachability graph defined by the stubborn sets chosen at each marking. The remaining problem is to compute

$$R' : (Y = X^\circ \cap W) \wedge P,$$

starting from

$$Q' : \emptyset \neq X \subseteq G \wedge P,$$

that is, $(Y = X^\circ \setminus (G \uplus B)) \wedge P$.

The algorithm we shall give is based on the idea of running an algorithm very much like the one described above to compute a stubborn set at *one* marking with the *whole* set X instead. First, all the dead markings are removed from the set X. This can be done, since the subset of markings M that do not enable any transitions, say A, can be precomputed for a safe Petri net. The intersection of A and X gives the dead markings in X. If the rest of X is empty, the algorithm is not applied. When the original algorithm to compute a stubborn set at a marking is to determine whether a transition is enabled at the given marking, the set under consideration (originally $X \setminus A$) is split into those that enable the marking, and to those that do not. The splitting guarantees that the enabledness of all the transitions that have caused a split to occur is well-defined for the set under consideration. Moreover, a split occurs only if the search for a stubborn set requires it.

If one of the parts resulting from the split is empty, it is ignored, since the empty set of markings has no followers and does not contribute to the result; both parts cannot be empty, since $X \setminus A$ was non-empty originally and empty sets are ignored. If both sets are non-empty, one is set aside with sufficient information on the state of the search to be able to resume the search later, and the processing of the other part is continued. Once a search terminates, a set is chosen from those set aside, and the state of the search associated with the set is restored, and the search resumed. This is continued until there are no more sets set aside. The set of follower markings of a set is produced by firing the enabled transitions in the stubborn sets found for the set. The set Y to be computed is then the union of the sets of white follower markings of all the split parts for which a stubborn sets was found. There will be a stubborn set for all the non-empty parts of $X \setminus A$, since if there is no stubborn set for some part, then this has no enabled transitions since T is stubborn otherwise. This contradicts the construction of A.

There are some complications involved, however. The enabledness of the transitions is not the only marking-dependent aspect of the computation of the stubborn sets using the given algorithm: also the choice of the scapegoat is marking dependent. Any transition with at least two input places can have two markings, both of which disable the transition, and the place that has too few tokens is unique but different for both, making the scapegoat different for these, see Figure 1 where the scapegoat depends whether the marking with the black or the striped tokens only is considered. A simple solution to the problem is to split the set under consideration further, setting aside parts in the exactly same manner if necessary. The other problem is that since the search for a stubborn set has not started from a transition that is enabled for all the markings in the set $X \setminus A$, there is no guarantee that the search will find a stubborn set, i.e. one with an enabled transition. Since all the markings enable at least one transition, any transition that has not been assigned a depth-first number in the search already can be used to continue the search.

We give the algorithm for this in Appendix B. Whenever the algorithm of Appendix A would compute the set $(t, m)^\circ$ for some transition t, the set of markings needs potentially to be split, sometimes to many pieces due to

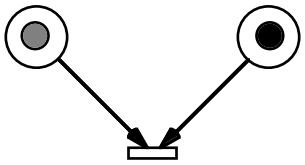

Figure 1: Ambiguity of the scapegoat.

the selection of the scapegoat. The fact that the algorithm of Appendix A is iterative is very useful here, since the state of the program is laid out in the form of explicit variables. The information to be set aside is the state of the search, basically all of the variables of the algorithm of Appendix A, with the notable exception of Y. There will not be a unique marking m, of course. Naturally, the variables that need not be implemented explicitly, like the set W, need not actually be set aside. The program will then repeat until the data structure used to set aside is empty.

It should be noted that the given algorithm only produces the set of reachable markings. It is fairly simple to also maintain a set of arcs between the markings produced.

5 Symmetry equivalence search

A symmetry is an *automorphism* on the unfolding of the Coloured Petri net: it is a bijection σ that maps places to places and transitions to transitions, and respects the weight function:

$$W(x, x') = W(\sigma(x), \sigma(x'))$$

for any $\langle x, x' \rangle \in (S \cup T) \times (T \cup S)$ in the unfolding. Obviously, a symmetry $\sigma : P \to P$ can be extended to a function on markings $\sigma : (P \to \mathsf{N}) \to (P \to \mathsf{N})$ by setting $(\sigma(m))(p) = m(\sigma(p))$. The crucial property of any symmetry σ is that

$$m\ [t\rangle\ m' \Rightarrow \sigma(m)\ [\sigma(t)\rangle\ \sigma(m')$$

for any $m, m' \in \mathsf{M}$, and $t \in T$. Since the symmetries are bijective, they are invertible, making the implication above in fact an equivalence.

We shall consider a given set of symmetries $\sum$ that includes the identity function and the inverse function of any of its elements, and which is closed with respect to functional composition. Such a set forms a *group* with the identity function as the identity element and functional composition as the binary operation, as can be easily verified. Based on the set $\sum$ being a group, we can define an equivalence relation $\simeq$ on markings by setting

$$m \simeq m' = (\exists \sigma : \sigma \in \sum : m = \sigma(m')).$$

A marking m is called *symmetric* iff $(\forall \sigma : \sigma \in \sum : m = \sigma(m))$. A symmetric marking forms a block of the equivalence relation $\simeq$ by itself, as any marking equivalent to it will also be equal to it. These definitions have followed the presentation in [19]. Given a set of initial markings M, a $\simeq$-reachablity graph is then a graph $\langle \tilde{V}, \tilde{A} \rangle$ such that

$$\begin{aligned} S\langle \tilde{V}, \tilde{A} \rangle : M \subseteq \tilde{V} \subseteq V \wedge (\forall m : m \in \tilde{V} : (\exists m_0 : m_0 \in M : m_0 \left[\tilde{V}\right\rangle^* m)) \\ \wedge (\forall m, m' : m, m' \in \tilde{V} \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\ \wedge (\forall m, m', t : m, m' \in \tilde{V} \wedge t \in T \wedge m\ [t\rangle\ m' : (\exists m'' : m'' \in \tilde{V} : m'' \simeq m')) \\ \wedge \tilde{A} = \{\langle m, t, m' \rangle \mid m, m' \in \tilde{V} \wedge t \in T \wedge (\exists m'' : m\ [t\rangle\ m'' : m' \simeq m'')\}. \end{aligned}$$

Basically, this is graph of connected representatives of the blocks of the equivalence relation $\simeq$: there is exactly one for each block unless the initial markings contain equivalent markings, all are reachable from some initial marking, and the arcs are the ones induced between the blocks.

We shall extend the framework of Section 3 to compute a $\simeq$-reachability graph. The goal is

$$R : S\langle B, E \rangle.$$

Let us refine the partition of the set of reachable markings to

$$P_0 : V = B \uplus G \uplus W \uplus C,$$

where C is the set of "clear" or ignored markings. We shall also use the invariant

$$P_1 : (\forall w : w \in W : (\exists g : g \in G : g\ [W\rangle^*\ w)).$$

Let us weaken the goal: we shall consider it in smaller conjuncts for the sake of clarity.

$$P_2 : M \subseteq B \cup G \subseteq V,$$

$$P_3 : (\forall m : m \in B \cup G : (\exists m_0 : m_0 \in M : m_0\ [B \cup G\rangle^*\ m)),$$

$$P_4 : (\forall m, m' : m, m' \in B \cup G \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m'),$$

$$P_5 : E = \{\langle m, t, m'\rangle \mid m \in B \wedge m' \in B \cup G \wedge t \in T \wedge (\exists m'' : m\ [t\rangle\ m'' : m' \simeq m'')\}$$

We shall also need to describe the function of the clear block of the partition:

$$P_6 : (\forall m : m \in C : (\exists b : b \in B \cup G : b \simeq m)).$$

Let us call the whole invariant

$$P : P_0 \wedge P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5 \wedge P_6.$$

Now,

$$\begin{array}{ll} & P \wedge G = \emptyset \\ = & P_0 \wedge P_1 \wedge G = \emptyset \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5 \wedge P_6 \\ = & P \wedge W = G = \emptyset \\ \Rightarrow & P \wedge W = G = \emptyset \wedge V = B \uplus C \\ \Rightarrow & \{(*)\} \\ & R, \end{array}$$

where the major point in (*) is that any successor of a black marking b is either black or clear, $b^\bullet \subseteq V = B \uplus C$, so by P_6 the missing conjunct of R is implied. The initialization is

$$\begin{array}{ll} & \wp(\text{“}B, G, W, E, C := \emptyset, M, V \setminus M, \emptyset, \emptyset\text{”}, P) \\ = & V = \emptyset \uplus M \uplus (V \setminus M) \uplus \emptyset \\ & \wedge (\forall w : w \in V \setminus M : (\exists g : g \in M : g\ [V \setminus M\rangle^*\ w)) \\ & \wedge M \subseteq \emptyset \cup M \subseteq V \\ & \wedge (\forall m : m \in \emptyset \cup M : (\exists m_0 : m_0 \in M : m_0\ [\emptyset \cup M\rangle^*\ m)) \\ & \wedge (\forall m, m' : m, m' \in \emptyset \cup M \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\ & \wedge \emptyset = \{\langle m, t, m'\rangle \mid m \in \emptyset \wedge \ldots\} \wedge (\forall m : m \in \emptyset : \ldots) \\ = & M \subseteq V \wedge (\forall w : w \in V : (\exists g : g \in M : g\ [\rangle^*\ w)) \\ & \wedge (\forall m : m \in M : (\exists m_0 : m_0 \in M : m_0\ [M\rangle^*\ m)) \\ & \wedge (\forall m, m' : m, m' \in M \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : \ldots) \\ & \wedge \emptyset = \{\langle m, t, m'\rangle \mid m \in \emptyset \wedge \ldots\} \wedge (\forall m : m \in \emptyset : \ldots) \\ = & M \subseteq V \wedge (\forall w : w \in V : (\exists m_0 : m_0 \in M : m_0\ [\rangle^*\ w)) \\ \Leftarrow & Q, \end{array}$$

where Q has been given in Section 3.

5.1 Progress

For the progress we consider first proceeding the same way as in Section 3. The invariant P_1 has already been considered there. For the rest, we compute

$$\begin{array}{ll} & \wp(\text{“}B, G, W := B \cup X, (G \setminus X) \cup (X^\bullet \cap W), W \setminus X^\bullet\text{”}, P_0) \\ = & V = (B \cup X) \uplus (G \setminus X) \cup (X^\bullet \cap W) \uplus W \setminus X^\bullet \uplus C \\ \Leftarrow & P_0 \wedge \emptyset \neq X \subseteq G, \\ & \wp(\text{“}B, G, W := B \cup X, (G \setminus X) \cup (X^\bullet \cap W), W \setminus X^\bullet\text{”}, P_2) \\ = & M \subseteq B \cup X \cup (G \setminus X) \cup (X^\bullet \cap W) \subseteq V \\ \Leftarrow & \{X \subseteq G \Rightarrow G = X \cup (G \setminus X)\} \\ & X \subseteq G \wedge M \subseteq B \cup G \cup (X^\bullet \cap W) \subseteq V \\ \Leftarrow & \{P_0 \Rightarrow W \subseteq V\} \\ & P_0 \wedge P_2 \wedge X \subseteq G, \\ & \wp(\text{“}B, G, W := B \cup X, (G \setminus X) \cup (X^\bullet \cap W), W \setminus X^\bullet\text{”}, P_3) \end{array}$$

$= (\forall m : m \in B \cup X \cup (G \setminus X) \cup (X^\bullet \cap W) :$
$\quad (\exists m_0 : m_0 \in M : m_0\ [B \cup X \cup (G \setminus X) \cup (X^\bullet \cap W)\rangle^*\ m))$
$\Leftarrow$ {$X \subseteq G \Rightarrow G = X \cup (G \setminus X)$}
$X \subseteq G$
$\wedge\ (\forall m : m \in B \cup G \cup (X^\bullet \cap W) : (\exists m_0 : m_0 \in M : m_0\ [B \cup G \cup (X^\bullet \cap W)\rangle^*\ m))$
$\Leftarrow$ {splitting}
$X \subseteq G$
$\wedge(\forall m : m \in B \cup G : (\exists m_0 : m_0 \in M : m_0\ [B \cup G \cup (X^\bullet \cap W)\rangle^*\ m))$
$\wedge(\forall m : m \in X^\bullet \cap W : (\exists m_0 : m_0 \in M : m_0\ [B \cup G \cup (X^\bullet \cap W)\rangle^*\ m))$
$\Leftarrow$ {(**)}
$P_3 \wedge X \subseteq G,$

where (**) is proven by noting that $P_3 \wedge B \cup G \subseteq B \cup G \cup (\ldots)$ implies the first universal quantification, and that the second follows from the first and $X \subseteq G$ since there is always a path of length one from any x to any $y \in x^\bullet$.

$\wp(\text{“}B, G, W := B \cup X, (G \setminus X) \cup (X^\bullet \cap W), W \setminus X^\bullet\text{”}, P_4)$
$= (\forall m, m' : m, m' \in B \cup X \cup (G \setminus X) \cup (X^\bullet \cap W)$
$\quad \wedge\ m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m')$
$\Leftarrow$ {$X \subseteq G \Rightarrow G = X \cup (G \setminus X)$}
$X \subseteq G \wedge (\forall m, m' : m, m' \in B \cup G \cup (X^\bullet \cap W)$
$\quad \wedge\ m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m')$
$\Leftarrow$ {see Appendix C}
$P_0 \wedge P_4 \wedge X \subseteq G \wedge R_0 \wedge R_1$

where

$$R_0 : (\forall m, m' : m \in B \cup G \wedge m' \in X^\bullet \cap W : m \not\simeq m')$$

and

$$R_1 : (\forall m, m' : m, m' \in X^\bullet \cap W \wedge m \simeq m' : m = m').$$

The predicates R_0 and R_1 are clearly not implied by P, but they depend only on the elements of known sets $X^\bullet$, B, and G. Consider then

$\wp(\text{“}B, G, W := B \cup X, (G \setminus X) \cup (X^\bullet \cap W), W \setminus X^\bullet\text{”}, P_5)$
$= E = \{\langle m, t, m'\rangle \mid m \in B \cup X \wedge m' \in B \cup X \wedge t \in T \wedge (\exists m'' : m\ [t\rangle\ m'' : m' \simeq m'')\}$
$=$ {splitting the quantification ranges}
$E = \{\langle m, t, m'\rangle \mid m \in B \wedge m' \in B \wedge t \in T \wedge (\exists m'' : m\ [t\rangle\ m'' : m' \simeq m'')\}$
$\quad \cup\ \{\langle m, t, m'\rangle \mid m \in X \wedge m' \in B \cup X \wedge t \in T \wedge (\exists m'' : m\ [t\rangle\ m'' : m' \simeq m'')\}.$

Clearly, the first set is the value of E implied by P_5. The latter set can be computed by deciding for each of the nodes in $X^\bullet$, whether there is a black marking equivalent to it.

Let us then treat P_6:

$\wp(\text{“}B, G, W := B \cup X, (G \setminus X) \cup (X^\bullet \cap W), W \setminus X^\bullet\text{”}, P_6)$
$= (\forall m : m \in C : (\exists b : b \in B \cup X \cup (G \setminus X) \cup (X^\bullet \cap W) : b \simeq m))$
$\Leftarrow$ {$X \subseteq G \Rightarrow G = X \cup (G \setminus X)$}
$X \subseteq G \wedge (\forall m : m \in C : (\exists b : b \in B \cup G \cup (X^\bullet \cap W) : b \simeq m))$
$\Leftarrow X \subseteq G \wedge (\forall m : m \in C : (\exists b : b \in B \cup G : b \simeq m))$
$= X \subseteq G \wedge P_6.$

The remaining problem is, how to establish the predicates R_0 and R_1 given $\emptyset \neq X \subseteq G \wedge P$. Perhaps the simplest strategy is to look at each marking $m' \in X^\bullet$, one at a time, and decide its equivalence: if m' is equivalent to a black or grey marking, or a white one in $X^\bullet$ treated prior to it, m' will be colored clear. This will also maintain P_6: all clear markings will, after making X black and $X^\bullet \cap W$ grey, have an equivalent black or grey marking. Since the identity function is a symmetry, $m = m' \Rightarrow m \simeq m'$, or equivalently, $m \not\simeq m' \Rightarrow m \neq m'$, it is not necessary to separately check whether m' is grey or black.

Collecting the results of the discussion, we get

$$
\begin{array}{l}
\{Q\}\ \ B, G, W, C, E := \emptyset, M, V \setminus M, \emptyset, \emptyset \{P\} \\
;\ \ \textbf{do}\ \ G \neq \emptyset \rightarrow x := pick(G) \{P \wedge x \in G\} \\
\ \ ; t, S := \bot, \emptyset \\
\ \ ;\ \ \textbf{do}\ \ t \neq \infty \rightarrow \\
\qquad \textbf{if}\ \ x\ [t\rangle\ \rightarrow y := nextmark(x, t) \\
\qquad\ ; S' := [y] \cap (B \cup G \cup S) \\
\qquad\ ;\ \ \textbf{if}\ \ S' \neq \emptyset \rightarrow \\
\qquad\qquad \textbf{do}\ \ S' \neq \emptyset \rightarrow z := pick(S') \\
\qquad\qquad ; \textbf{if}\ \ z \in B \rightarrow E := E \cup \{\langle x, t, z\rangle\} [] z \notin B \rightarrow \textbf{skip}\ \textbf{fi} \\
\qquad\qquad ; S' := S' \setminus \{z\} \\
\qquad\qquad\ \ \textbf{od} \\
\qquad\ ; C := C \cup \{y\} \\
\qquad\qquad [] S' = \emptyset \rightarrow S, W := S \cup \{y\}, W \setminus \{y\} \\
\qquad\qquad\ \ \textbf{fi} \\
\qquad\ [] \neg x\ [t\rangle\ \rightarrow \textbf{skip} \\
\qquad\ \ \textbf{fi} \\
\ \ ; t := nexttrans(t) \\
\qquad \textbf{od} \\
;\ B, G := B \cup \{x\}, (G \setminus \{x\}) \cup S \\
\qquad \textbf{od} \\
\{P \wedge G = \emptyset\}\ \ \{R\}
\end{array}
$$

where *pick* now returns any one element of its argument set.

This strategy has the drawback of making the generation of the $\simeq$-reachability graph a marking-at-a-time process. This denies, at least to a degree, the benefits of the use of efficient representations for sets of states, such as the BDDs. The *compatible projection operator* proposed by Lin and Newton [14, 13] offers a chance to improve upon this. The operator can be used to compute a function that maps each marking to a unique representative of its equivalence class in an equivalence relation, such as the symmetry relation $\simeq$ on markings is, given in a BDD form. Since this can be precomputed for the symmetry relation of a safe Petri net (or one with known bounds on markings of all the places), the unique representative function can be precomputed and represented in BDD form, and can be applied to achieve R_0 and R_1. Obviously, the set of all reachable markings—and not only those found during the symmetry search—can also be computed given a BDD representation of the symmetry relation and the $\simeq$-reachability graph, the relation can be applied to the set of all markings in the graph, which can, if wanted, be also maintained in BDD form.

5.2 Discussion

We have presented a concise derivation of the algorithm to compute the $\simeq$-reachability graph with no restrictions placed on the symmetries. [11] considers only symmetries that are generated by maps of colors of a Coloured Petri net: any symmetry is supposed to map a place-color pair $\langle p, c\rangle$ or a transition-binding pair $\langle t, b\rangle$ to another only if the first components are equal. The approach taken in [19, 18] and here is slightly more general, since this restriction is not placed on the symmetries. Though here we have assumed that the symmetries are given, [18] considers an algorithm to compute the set of symmetries of a Petri net.

The $\simeq$-reachability graph provides only a non-reachability test if the initial marking considered is not symmetric: otherwise, given a sequence $\tilde{\sigma}$ in the $\simeq$-reachability graph there will be a corresponding sequence σ in the ordinary reachability graph, but it is only guaranteed that the initial marking of σ is *equivalent* to that of $\tilde{\sigma}$. For a symmetric initial marking this implies equality, otherwise not necessarily so. If the set of initial markings M is closed with respect to all the symmetries $\sigma \in \sum$, then the reachability test is again valid, since then there is an initial marking from which the marking is reachable from.

Combining the symmetry and stubborn set methods turns out to be trivial: the only modification caused by the stubborn set method to the above is the use of the stubborn set related notation, that is, $(\ldots)^\circ$ instead of $(\ldots)^\bullet$, $m\ [(t)\rangle_s$ instead of $m\ [t\rangle$, and, say, $nexttrans_s$ instead of $nexttrans$, and $nextmark_s$ instead of $nextmark$.

Finally, it should be noted that dropping P_4 and R_1 in order to be able to use the BDD representation of states to its fullest will make the use of symmetries superfluous. The point of using the symmetries is to avoid having multiple representatives of the equivalence classes in the $\simeq$-reachability graph. We have discussed using

the *compatible projection operator* of Lin and Newton and an explicit representation of the symmetry equivalence relation in the form of a BDD to benefit in the symmetry equivalence search, however.

6 Conclusion

We have defined the concept of a generalized reachability graph of a Petri net that can potentially be used to benefit in analyzing the behavior of fault-tolerant systems, and discussed a symbolic, set-at-a-time algorithm for the generation of such a graph. We have also discussed the generalization of one of Valmari's stubborn set selection algorithms to operate on a set of markings at a time. The stubborn sets offer a chance of reducing the number of interleavings of transitions considered during the symbolic that still is a problem for the symbolic search for asynchronous systems [6]. The use of BDDs in these algorithms extends the field of application of this important data structure, used to much benefit in hardware verification, to software verification, among others. This extension is achieved through the use of Petri nets as the basic system model, since Petri nets have been widely applied, and not only within computer science. Combining the two methods offers then a chance for increased applicability of reachability based analysis methods by potentially increasing the size of the models that can be analysed.

More recently, Valmari has also presented an algorithm that verifies any pregiven stuttering-invariant linear time temporal logic formula [20], and then an on-the-fly verification algorithm [23], both using stubborn sets. Relating the other definitions of stubborns sets and these to the symbolic search is left for future work. A specific problem that remains for future work also is how to coordinate the nondeterministic choices appearing in the algorithms to achieve a significant extension of the applicability of the methods. These choices are picking the ordering the variables, picking X, picking transitions and scapegoats during the stubborn set computation, and picking which part to shelve. For choosing the ordering of the variables, there has been work done to develop heuristics based on the application to hardware verification. These can form a basis for the considerations involved.

We have also discussed in more detail than previously combining the symmetry equivalence search with the stubborn set method. The methods turned out to be compatible in that they operate on "orthogonal" aspects of the problem. Moreover, we have discussed the use of BDDs in the symmetry equivalence search. In the context of temporal logic the same ideas have been put forward by Clarke et al [8]. The use of the compatible projection operator of Lin and Newton has not been exploited so far in the symmetry equivalence search.

Acknowledgements

This work has been supported by the National Science Foundation, Finnish Academy, the Finnish Cultural Foundation (Kalle and Dagmar Välimaa fund), the Emil Aaltonen Foundation, the Alfred Kordelin Foundation, the Information Processing Research Foundation (Tietotekniikan tutkimussäätiö, and by the Helsinki University of Technology directly, through Ernst Wirtzen fund, and through the Foundation for Financial Support of HUT.
Ps 127.

References

[1] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C–27:509–516, June 1978.

[2] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Applications and Relationships to Other Models of Concurrency, Part I*, volume 254 of *Lecture Notes in Computer Science*, Berlin, Germany, 1987. Springer-Verlag. Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, Germany, September 1986.

[3] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Applications and Relationships to Other Models of Concurrency, Part II*, volume 255 of *Lecture Notes in Computer Science*, Berlin, Germany, 1987. Springer-Verlag. Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, Germany, September 1986.

[4] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C–35(6):677–691, 1986.

[5] R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[6] J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. In A. Halaas and P. B. Denyer, editors, *Proceedings of the 1991 International Conference on Very Large Scale Integration*, Aug. 1991.

[7] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Fifth Annual IEEE Symposium on Logic in Computer Science*. IEEE, June 1990.

[8] E. M. Clarke, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. In Courcoubetis [9], pages 450–462.

[9] C. Courcoubetis, editor. *Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[10] E. W. D. Dijkstra. Guarded commands, nondeterminacy and the formal derivation of programs. *Communications of the ACM*, 18:453–457, Aug. 1975.

[11] P. Huber, A. M. Jensen, L. O. Jepsen, and K. Jensen. *Towards Reachability Trees for High-Level Petri Nets*, volume 188 of *Lecture Notes in Computer Science*, pages 215–233. Springer-Verlag, Berlin, Germany, 1985.

[12] C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell Systems Technical Journal*, 38:985–999, July 1959.

[13] B. Lin and A. R. Newton. Efficient symbolic manipulation of equivalence relations and classes. In *Proceedings of Formal Methods in VLSI Design Workshop, Miami, FL, January 1991*, 1991.

[14] B. Lin and A. R. Newton. Implicit manipulation of equivalence classes using binary decision diagrams. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors, Cambridge, MA, October 14–16, 1991. IEEE Computer Society Press, Los Alamitos, CA 1991, 654 p.*, page 6, 1991.

[15] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[16] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, Germany, 1962. Schriften des IIM Nr. 2. English translation: [17].

[17] C. A. Petri. Communication with automata. Technical Report RADC–TR–65–377, Griffith Air Force Base, New York, 1966.

[18] K. Schmidt and P. H. Starke. An algorithm to compute the symmetries of Petri nets. *Petri Net Newsletter*, 40:25–30, Dec. 1991.

[19] P. H. Starke. Reachability analysis of Petri nets using symmetries. *Systeme: Analysis, Modellierung, Simulation*, 8(4/5):293–303, 1991.

[20] A. Valmari. Stubborn attack on state explosion. *Formal Methods in System Design*, 1:297–322, 1991.

[21] A. Valmari. *Stubborn Sets for Reduced State Space Generation*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer-Verlag, Berlin, Germany, 1991.

[22] A. Valmari. Stubborn sets of coloured Petri nets. In *Papers Presented at the 12th International Conference on Application and Theory of Petri Nets*, pages 102–121, Gjern, Denmark, June 1991. Åarhus University.

[23] A. Valmari. On-the-fly verification with stubborn sets. In Courcoubetis [9], pages 397–408.

A An iterative stubborn set algorithm

The following algorithm computes the stubborn postset of a single marking m, $Y = \{m\}^\circ$. It will not search for a transition that is enabled at m, but will, if necessary go through them all. The variable e is used to terminate the search once a suitable strongly connected component (SCC) has been found. The symbol $\perp$ denotes the smallest transition in a given but arbitrary linear order on the transitions; we have assumed there to be at least one transition in the net under consideration.

$$
\begin{array}{l}
\quad B, t, e := \emptyset, \bot, \textbf{false} \\
;\ \ \textbf{do}\ \neg e \wedge t \neq \infty \rightarrow \\
\quad \textbf{if}\ t \in B \rightarrow \textbf{skip} \\
\quad [\!]\, t \notin B \rightarrow B, G, W, d(t), q, s(0), s(1), s(2), f(t), r, h(0), S := \\
\qquad B \cup \{t\}, (t,m)^\circ \setminus \{t\}, T \setminus ((t,m)^\circ \cup B \cup \{t\}), |B|, 2, \emptyset, t, (t,m)^\circ \setminus \{t\}, |B|, 1, t, \{t\} \\
\quad ;\ \ \textbf{do}\ q \neq 0 \rightarrow \mathit{EXTEND} \\
\quad ;\ \ \textbf{do}\ q \neq 0 \ \text{cand}\ s(q+1) \in B \rightarrow \\
\quad \textbf{if}\ d(s(q-1)) > d(s(q+1) \wedge s(q+1) \in S \rightarrow \\
\qquad f(s(q-1)) := f(s(q-1)) \min f(s(q+1)) \\
\quad [\!]\, d(s(q-1)) \leq d(s(q+1) \vee s(q+1) \notin S \rightarrow \textbf{skip} \\
\quad \textbf{fi} \\
\quad ; \mathit{EXTEND} \\
\quad \textbf{od} \\
\quad ; \textbf{if}\ q = 0 \rightarrow \textbf{skip} \\
\quad [\!]\, q \neq 0 \rightarrow B, G, W, d(s(q-1)), q, s(q+2), f(s(q-1)), r, h(r), S := \\
\qquad B \cup \{s(q-1)\}, (G \setminus \{s(q-1)\}) \cup ((s(q-1),m)^\circ \cap W), \\
\qquad W \setminus (s(q-1),m)^\circ, |B|, q+2, (s(q-1),m)^\circ \setminus \{s(q-1)\}, \\
\qquad |B|, r+1, s(q+1), S \cup \{s(q+1)\} \\
\quad \textbf{fi} \\
\quad \textbf{od} \\
\quad \textbf{fi} \\
;\ t := \mathit{nexttrans}(t) \\
\quad \textbf{od}
\end{array}
$$

where *EXTEND* is the program

$$
\begin{array}{l}
\quad \textbf{do}\ q \neq 0 \ \text{cand}\ s(q) = \emptyset \rightarrow \\
\quad \textbf{if}\ f(s(q-1)) = d(s(q-1)) \rightarrow \{\text{found root of an SCC}\} \\
\quad \textbf{do}\ h(r) \neq s(q-1) \rightarrow \\
\quad \textbf{if}\ m\ [h(r)\rangle\ \rightarrow e, Y := \textbf{true}, Y \cup \{\mathit{nextmark}(m, h(r))\} \\
\quad [\!]\, \neg m\ [h(r)\rangle\ \rightarrow \textbf{skip} \\
\quad \textbf{fi} \\
\quad ; f(h(r)), r, S := d(s(q-1)), r-1, S \setminus \{h(r)\} \\
\quad \textbf{od} \\
\quad ; r, S := r-1, S \setminus \{h(r)\} \\
\quad [\!]\, f(s(q-1)) \neq d(s(q-1)) \rightarrow \textbf{skip} \\
\quad \textbf{fi} \\
\quad ; \textbf{if}\ e \rightarrow q := 0 [\!]\, \neg e \rightarrow q := q-2\ \textbf{fi} \\
\quad \textbf{od} \\
;\ \textbf{if}\ q = 0 \rightarrow \textbf{skip} [\!]\, q \neq 0 \rightarrow s(q+1) := \mathit{pick}(s(q)); s(q) := s(q) \setminus \{s(q+1)\}\ \textbf{fi}\ .
\end{array}
$$

Note that $s(q)$ is maintained as the *set* of followers of the transition according to the definition of stubborn sets. This makes the search of a follower transition easier to express than the search of a follower marking is. We have taken the liberty of using the notation $(t,m)^\circ$ of this set of transitions. This makes it easier to relate the algorithm to an iterative version of Tarjan's algorithm to find the fragments of a graph.

B A symbolic stubborn set algorithm

The following algorithm computes the stubborn postset Y of a set of markings X, $Y = X^\circ$. The variable e is used to terminate the search once a suitable strongly connected component (SCC) has been found. The symbol $\bot$ denotes the smallest transition in a given but arbitrary linear order on the transitions; we have assumed there to be at least one transition in the net under consideration. We assume $A \subseteq \mathsf{M}$ is the set of markings that enable no transition of the net.

$D, X, Y := X \cap A, X \setminus A, \emptyset$; **if** $D = \emptyset \to$ **skip** $[]\ D \neq \emptyset \to$ "report D as deadlocks" **fi**
; **if** $X = \emptyset \to$ **skip**
$[]\ X \neq \emptyset \to B, t, e := \emptyset, \bot,$ **false**
; **do** $\neg e \wedge t \neq \infty \to$
if $t \in B \to$ **skip**
$[]\ t \notin B \to$ *SPLIT1*
; **do** $q \neq 0 \to$ *EXTEND*
; **do** $q \neq 0$ cand $s(q+1) \in B \to$
if $d(s(q-1)) > d(s(q+1)) \wedge s(q+1) \in S \to$
$f(s(q-1)) := f(s(q-1)) \min f(s(q+1))$
$[]\ d(s(q-1)) \leq d(s(q+1)) \vee s(q+1) \notin S \to$ **skip**
fi
; *EXTEND*
od
; **if** $q = 0 \to$ **skip** $[]\ q \neq 0 \to$ *SPLIT2* **fi**
od
fi
; **if** $e \to$ *UNSHELVE*$(X, p, B, G, W, t, e, d, q, s, f, r, h, S, E)$ $[]\ \neg e \to t := nexttrans(t)$ **fi**
od
fi

where *EXTEND* is the program

do $q \neq 0$ cand $s(q) = \emptyset \to$
if $f(s(q-1)) = d(s(q-1)) \to$ {found root of an SCC}
do $h(r) \neq s(q-1) \to$
if $h(r) \in E \to e, Y :=$ **true**, $Y \cup \{nextmark(m, h(r)) \mid m \in X\}$
$[]\ h(r) \notin E \to$ **skip**
fi
; $f(h(r)), r, S := d(s(q-1)), r-1, S \setminus \{h(r)\}$
od
; $r, S := r-1, S \setminus \{h(r)\}$
$[]\ f(s(q-1)) \neq d(s(q-1)) \to$ **skip**
fi
; **if** $e \to q := 0$ $[]\ \neg e \to q := q-2$ **fi**
od
; **if** $q = 0 \to$ **skip** $[]\ q \neq 0 \to s(q+1) := pick(s(q)); s(q) := s(q) \setminus \{s(q+1)\}$ **fi** ,

SPLIT1 is the program

$B, G, W, d(t), q, s(0), s(1), f(t), r, h(0), S, E :=$
$B \cup \{t\}, \emptyset, T \setminus (B \cup \{t\}), |B|, 2, \emptyset, t, |B|, 1, t, \{t\}, \emptyset$
; $X, X', H := X \cap \{m \in \mathsf{M} \mid m\ [t\rangle\ \}, X \setminus \{m \in \mathsf{M} \mid m\ [t\rangle\ \}, {}^\bullet t$
; **do** $X' \neq \emptyset \to g := pick(H); H := H \setminus \{g\}$
$X', X'' := X' \setminus \{m \in \mathsf{M} \mid m(g) < W(g,t)\}, X' \cap \{m \in \mathsf{M} \mid m(g) < W(g,t)\}$
; $s(q) := {}^\bullet g \setminus \{t\}$; *SHELVE*$(X'', B, {}^\bullet g \setminus \{t\}, W \setminus {}^\bullet g, t, e, d, q, s, f, r, h, S, E)$
od
; **if** $X = \emptyset \to$
UNSHELVE$(X, B, G, W, t, e, d, q, s, f, r, h, S, E)$
$[]\ X \neq \emptyset \to$
$G, W, s(q), E := ({}^\bullet t)^\bullet \setminus \{t\}, W \setminus ({}^\bullet t)^\bullet, ({}^\bullet t)^\bullet \setminus \{t\}, E \cup \{t\}$
fi ,

and *SPLIT2* is the program

$$
\begin{array}{l}
\quad B, G, d(s(q-1)), q, f(s(q-1)), r, h(r), S := \\
\qquad B \cup \{s(q-1)\}, G \setminus \{s(q-1)\}, |B|, q+2, |B|, r+1, s(q+1), S \cup \{s(q+1)\} \\
\quad ; X, X', H := X \cap \{m \in \mathsf{M} \mid m\ [s(q-1)\rangle\ \}, X \setminus \{m \in \mathsf{M} \mid m\ [s(q-1)\rangle\ \}, {}^\bullet s(q-1) \\
; \quad \textbf{do}\ \ X' \neq \emptyset \to g := pick(H); H := H \setminus \{g\} \\
\quad X', X'' := X' \setminus \{m \in \mathsf{M} \mid m(g) < W(g, s(q-1))\}, X' \cap \{m \in \mathsf{M} \mid m(g) < W(g, s(q-1))\} \\
\quad ; \mathit{SHELVE}(X'', B, {}^\bullet g \setminus \{s(q-1)\}, W \setminus {}^\bullet g, t, e, d, q, s, f, r, h, S, E) \\
\quad \textbf{od} \\
; \textbf{if}\ \ X = \emptyset \to \mathit{UNSHELVE}(X, B, G, W, t, e, d, q, s, f, r, h, S, E) \\
\quad [] X \neq \emptyset \to G, W, s(q), E := ({}^\bullet t)^\bullet \setminus \{t\}, W \setminus ({}^\bullet t)^\bullet, ({}^\bullet t)^\bullet \setminus \{t\}, E \cup \{t\} \\
\quad \textbf{fi}
\end{array}
$$

Note that $s(q)$ is maintained as the *set* of followers of the transition according to the definition of stubborn sets. This makes the search of a follower transition easier to express than the search of a follower marking is. The programs *SPLIT1* and *SPLIT2* perform as much as possible of the assignments to the program variables as possible before actually splitting the set X. The programs *SHELVE* and *UNSHELVE* that have not been explicitly given are assumed to save the given values as a unit and restore them to the given variables, respectively. After a restoration, the unit used to restore the values of the variables is discarded. No assumption is made as to the order of the restoration of the saved units; it is assumed, however, that if there are no more units to use for restoration at the time *UNSHELVE* is called, this will return the values of **true** and 0 for e and q, respectively, to terminate the program.

C Symmetry Equivalence Search

This appendix presents in more detail a part of a proof appearing in Section 5 of the paper.

Let us recall the relevant definitions:

$$m \simeq m' = (\exists \sigma : \sigma \in \sum : m = \sigma(m')),$$

$$
\begin{array}{l}
S\langle \tilde{V}, \tilde{A} \rangle : M \subseteq \tilde{V} \subseteq V \wedge (\forall m : m \in \tilde{V} : (\exists m_0 : m_0 \in M : m_0 \left[\tilde{V}\right\rangle^* m)) \\
\quad \wedge (\forall m, m' : m, m' \in \tilde{V} \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\
\quad \wedge (\forall m, m', t : m, m' \in \tilde{V} \wedge t \in T \wedge m\ [t\rangle\ m' : (\exists m'' : m'' \in \tilde{V} : m'' \simeq m')) \\
\quad \wedge \tilde{A} = \{\langle m, t, m' \rangle \mid m, m' \in \tilde{V} \wedge t \in T \wedge (\exists m'' : m\ [t\rangle\ m'' : m' \simeq m'')\},
\end{array}
$$

$$R : S\langle B, E \rangle,$$

$$P_0 : V = B \uplus G \uplus W \uplus C,$$

where C is the set of "clear" or ignored markings,

$$P_1 : (\forall w : w \in W : (\exists g : g \in G : g\ [W\rangle^*\ w)),$$

$$P_2 : M \subseteq B \cup G \subseteq V,$$

$$P_3 : (\forall m : m \in B \cup G : (\exists m_0 : m_0 \in M : m_0\ [B \cup G\rangle^*\ m)),$$

$$P_4 : (\forall m, m' : m, m' \in B \cup G \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m'),$$

$$
\begin{array}{l}
P_5 : E = \{\langle m, t, m' \rangle \mid m \in B \wedge m' \in B \cup G \\
\qquad \wedge t \in T \wedge (\exists m'' : m\ [t\rangle\ m'' : m' \simeq m'')\},
\end{array}
$$

$$P_6 : (\forall m : m \in C : (\exists b : b \in B \cup G : b \simeq m)),$$

$$P : P_0 \wedge P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5 \wedge P_6,$$

$$R_0 : (\forall m, m' : m \in B \cup G \wedge m' \in X^\bullet \cap W : m \not\simeq m'),$$

and

$$R_1 : (\forall m, m' : m, m' \in X^\bullet \cap W \wedge m \simeq m' : m = m').$$

C.1 Progress on P_4

$$
\begin{aligned}
& \wp(\text{“}B, G, W := B \cup X, (G \setminus X) \cup (X^\bullet \cap W), W \setminus X^\bullet\text{”}, P_4) \\
= \; & (\forall m, m' : m, m' \in B \cup X \cup (G \setminus X) \cup (X^\bullet \cap W) \\
& \quad \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\
\Leftarrow \; & \{X \subseteq G \Rightarrow G = X \cup (G \setminus X)\} \\
& X \subseteq G \\
& \wedge (\forall m, m' : m, m' \in B \cup G \cup (X^\bullet \cap W) \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\
\Leftarrow \; & \{\text{splitting the quantification range}\} \\
& X \subseteq G \\
& \wedge (\forall m, m' : m, m' \in B \cup G \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\
& \wedge (\forall m, m' : m \in B \cup G \wedge m' \in X^\bullet \cap W \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\
& \wedge (\forall m, m' : m \in X^\bullet \cap W \wedge m' \in B \cup G \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\
& \wedge (\forall m, m' : m, m' \in X^\bullet \cap W \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\
\Leftarrow \; & P_4 \wedge X \subseteq G \\
& \wedge (\forall m, m' : m \in B \cup G \wedge m' \in X^\bullet \cap W \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\
& \wedge (\forall m, m' : m \in X^\bullet \cap W \wedge m' \in B \cup G \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\
& \wedge (\forall m, m' : m, m' \in X^\bullet \cap W \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\
\Leftarrow \; & \{\text{renaming quantification variables, } \simeq \text{ symmetric}\} \\
& P_4 \wedge X \subseteq G \\
& \wedge (\forall m, m' : m \in B \cup G \wedge m' \in X^\bullet \cap W \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\
& \wedge (\forall m, m' : m, m' \in X^\bullet \cap W \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\
\Leftarrow \; & \{\text{by } P_0\text{, a marking has exactly one color}\} \\
& P_0 \wedge P_4 \wedge X \subseteq G \\
& \wedge (\forall m, m' : m \in B \cup G \wedge m' \in X^\bullet \cap W \wedge (m \notin M \vee m' \notin M) : m \not\simeq m') \\
& \wedge (\forall m, m' : m, m' \in X^\bullet \cap W \wedge m \simeq m' \wedge (m \notin M \vee m' \notin M) : m = m') \\
\Leftarrow \; & \{P_0 \wedge P_2 \wedge w \in W \Rightarrow w \notin M\} \\
& P_0 \wedge P_2 \wedge P_4 \wedge X \subseteq G \\
& \wedge (\forall m, m' : m \in (B \cup G) \setminus M \wedge m' \in X^\bullet \cap W : m \not\simeq m') \\
& \wedge (\forall m, m' : m, m' \in X^\bullet \cap W \wedge m \simeq m' : m = m') \\
= \; & P_0 \wedge P_4 \wedge X \subseteq G \wedge R_0 \wedge R_1
\end{aligned}
$$