Automatic Symmetry Detection in Well-Formed Nets

Yann Thierry-Mieg, Claude Dutheillet, and Isabelle Mounier
(LIP6-SRC, France)

Laboratoire d'Informatique de Paris 6, France
Yann.Thierry-Mieg,Claude.Dutheillet,Isabelle.Mounier@lip6.fr

KEYWORDS: Well-Formed Petri nets, symmetry detection, symbolic model-checking, partial symmetry

Abstract. Formal verification of complex systems using high-level Petri Nets faces the so-called state-space explosion problem. In the context of Petri nets generated from a higher level specification, this problem is particularly acute due to the inherent size of the considered models. A solution is to perform a symbolic analysis of the reachability graph, which exploits the symmetry of a model.
Well-Formed Nets (*WN*) are a class of high-level Petri nets, developed specifically to allow automatic construction of a symbolic reachability graph (SRG), that represents equivalence classes of states. This relies on the definition *by the modeler* of the symmetries of the model, through the definition of "static sub-classes". Since a model is self-contained, these (a)symmetries are actually defined *by the model itself*.
This paper presents an algorithm capable of automatically extracting the symmetries inherent to a model, thus allowing its symbolic study by translating it to *WN*. The computation starts from the assumption that the model is entirely symmetric, then examines each component of a net to deduce the symmetry break it induces. This translation is transparent to the end-user, and is implemented as a service for the AMI-Net package. It is particularly adapted to models containing large value domains, yielding combinatorial gain in the size of the reachability graph.

1 Introduction

Formal verification of complex systems using high-level Petri nets (HLPN) faces the state-space size explosion problem. This is mainly due to interleaving between sequences of possible execution, and to the increase in the size of the variable domains. Lastly, and to a lesser extent, we have the impact of the number of tokens regardless of their value.

Well Formed nets (*WN*) [2] are a class of HLPN that allows automatic generation of a Symbolic Reachability Graph (SRG) [3]. An SRG allows compact representation of the state-space by defining classes of equivalent concrete accessible states. This is made possible by the correct definition *by the modeler* of the classes of structurally equivalent objects (tokens). In this paper, we will show that such a definition is not truly necessary, as it can be deduced from the model itself. The gains of the symbolic approach are tremendous, as it reduces the variable domain size to the lesser token

cardinality problem. Furthermore it leaves room to apply structural reductions to fight the interleaving problem [11, 4].

We propose here a syntactic sugar for the definition of *WN*, which does not require prior definition of the symmetries but allows to calculate them automatically. This is not an extension of the *WN* formalism, as we retain exactly the same expression power. Calculating the symmetry allowed by a given model automatically has several advantages:

- The fact that *symbolic* model-checking is being performed is transparent to the user. Therefore non-expert users may use the advantages of symbolic study more easily.
- Even expert users have a difficult time defining symmetries, and the syntax of *WN* is quite constraining in many respects. It forces to define all the elements of the net in terms of these equivalence classes. By allowing use of concrete predicates, we lighten the burden of the modeler.
- Although there is no risk of under-specification when defining a *WN*, there is a risk of over-specification. In other words, though one cannot put in the same equivalence class objects that have different structural behaviors, there is a risk of distinguishing between objects that are actually equivalent with respect to the model, or of defining operations like ordering that constrain the symmetry when they are not needed.
- Modifying the model is easier, for the same reasons.
- Moreover, if the net is being generated from a higher level specification language, such as **L*f*P**[12, 6], we require a process capable of detecting such symmetries without human intervention.

This work has been implemented as a tool in the CPN-AMI [1] software package, and relies on GreatSPN [8] of the university of Torino for the construction of the SRG. Beyond the context of this tool, the extraction of symmetries from the elements of a model has other applications being developed.

- It gives us insights on how to increase the symmetry of a model, by allowing to deduce which elements of the net are the most constraining on its symmetry.
- It allows, as an extension the ESRG technique [10], to take these constraints into account only when they are truly required. This allows symbolic study of only partially symmetric models.

The problem addressed here is also encountered in a variety of other situations where our technique might be successfully applied. It could be used to determine relevant limit values for automatic symbolic test-case generation such as in [13, 9]. It also might be used with other classes of models, such as the extensions to murphy [5] that take into account the same type of symmetries as *WN*.

This paper is organized as follows: section 2 introduces the *WN* formalism; section 3 defines the symmetries that are considered and their impact on the construction of a symbolic reachability graph; section 4 presents the syntactic extensions that are made to *WN* syntax, and their effect on the symmetry of the system; section 5 develops the rules that allow automatic symmetry extraction from a guard predicate; finally section 6 reports the performances of the tool implementing this technique on a realistic example.

2 Context

We recall here the main features of the WN model, and the way it can be used to automatically build a quotient graph, namely the Symbolic Reachability Graph. The construction of the SRG relies on the exploitation of symmetries that are explicitly contained in the model definition.

2.1 Well-Formed Nets: Definition

Well-formed Nets (WN) are a high-level Petri net model in which color domains and color functions must respect a simple, rigorous syntax, which automatically takes into account the symmetries of the system. WN tokens carry a composite information expressed as a tuple of colors (called objects), taken from possibly ordered *basic color classes*. Each color class represents system components of a given kind (e.g. the process class, the processor class,...).

If all the objects in a class do not behave the same way, the class must be partitioned into static subclasses. Objects in the same static subclass represent entities that always behave in a symmetric (homogeneous) way, while objects belonging to different static subclasses may have different behaviors. This constraint is reflected by the syntax of the color functions of the model, which makes it impossible to distinguish objects that belong to the same static subclass.

The general color functions of the WN model are built from three types of basic functions: the *projection* function, the *successor* function and the *diffusion/synchronization* function. The syntax used for the projection function is x, where x is one of the transition variables (i.e., one of the parameters that appears on an arc adjacent to the transition). It is called projection because it selects one element from the tuple of parameter values defining the transition color instance). The syntax used for the successor function is $!x$ where x is again one of the transition variables, it applies only to ordered classes and returns the successor of the color assigned to x in the transition color instance. Finally, the syntax for the diffusion/synchronization function is $C_i.all$ (or $C_i^j.all$): it is a constant function that returns the whole set of colors of class C_i (of static subclass $C_i^j \subset C_i$). It is called synchronization when used on a transition input arc because it implements a synchronization among a set of colored tokens contained into a place, while it is called diffusion when used on a transition output arc because it puts one token of each color of the (sub)class it applies to into a place.

Transitions can be associated with a predicate which restricts the set of their possible instantiations.

Definition 1. *Standard Predicates. A standard predicate (or guard) associated with a transition t is a boolean expression of* basic predicates*. The allowed basic predicates are: $x = y$, $x = !y$, $d(x) = C_i^j$, $d(x) = d(y)$, where $x,y \in Var_i(t)$ are parameters of t of the same type, $!y$ denotes the successor of y (assuming that the type of y is an ordered class), and $d(x)$ denotes the domain of x, which is the static subclass x belongs to.*

General color (arc) functions are defined as weighted sums of tuples. The elements composing the tuples are in turn weighted sums of *basic functions*, defined on basic color classes and returning multi-sets of colors in the same class.

Definition 2. *An arc function $\mathcal{F}$ associated with an arc connecting place p and transition t has the form:* $\mathcal{F} = \sum_k \alpha_k.F_k$ *where* α_k *is a positive integer and* F_k *is a function which maps an element of the color domain of transition t (cd(t)) onto a multi-set of colors of cd(p), color domain of p.*

Formally, $F_k : cd(t) \rightarrow Bag(cd(p))$ *is a function of the form*

$$F = \bigotimes_{C_i \in\ C} \bigotimes_{j=1,\ldots,e_i} f_i^j = \langle f_1^1, \ldots, f_1^{e_1}, \ldots, f_n^1, \ldots, f_n^{e_n} \rangle$$

with e_i *representing the number of occurrences of class* C_i *in color domain of place p, and* $\otimes$ *denoting the Cartesian product, i.e.,*
$cd(p) = \bigotimes_{C_i \in C} \bigotimes_{j=1,\ldots,e_i} C_i$.

Each function f_i^j *in turn is defined as:*

$$f_i = \sum_{q=1}^{ns_i} \beta_{i,q}.C_i^q.all + \sum_{x \in Var_i(t)} (\gamma_x.x + \delta_x.!x)$$

where $C_i^q.all$*, x and* $!x$ *are the diffusion, projection and successor functions,* $\beta_{i,q}$*,* γ_x *and* δ_x *are integer numbers taken such that the number of selected objects in a class is never negative.*

The syntax of a WN initial marking is quite similar to the syntax of color functions, except it does not use variables. The f_i are thus limited to the first term of the above equation when defining an initial marking. Although the syntax of color functions makes it possible to write complex expressions, the functions practically used in models are rather simple.

Example 1. Let *a* be an arc linking a place *p* of color domain $cd(p) = C_1 \times C_2$ with a transition *t* such that $cd(t) = C_1 \times C_1 \times C_2$, a possible color function of *a* is:
$\langle C_1.all - x, y \rangle$, where *x* and *y* are variables instantiated in C_1 and C_2 respectively. This function takes a set of tokens in *p*, composed of every element of C_1 except for one (the instance of *x*), and all having an identical associated value (the instantion of *y*). Let a' be another arc linking place p' to *t* such that $cd(p') = C_1 \times C_1$. A possible color function of a' is $\langle x \rangle + 2.\langle z \rangle$, where $x, z \in C_1$ are variables. This function requires three tokens in p' of which two must be identical.

The key consequence of these definitions is that the syntax of Well-formed Nets makes it *impossible* for objects belonging to the same static subclass to have different behaviors. Hence, the partition of color classes into static subclasses directly corresponds to the definition of the symmetries of the system. The complete definition of the model is summarized in the following paragraph.

Definition 3. *Well-formed Nets*

A Well-formed Net *is a seven-tuple:*

$$\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, C, cd, m_0 \rangle$$

where:

1. *P and T are disjoint finite non empty sets (the places and transitions of $\mathcal{N}$),*
2. *$C = \{C_1, \ldots, C_n\}$ is the finite set of finite* basic *color classes. Some of the color classes may be ordered.*
3. *cd is a function defining the color domain of each place and transition; for places it is expressed as cartesian product of classes of C (repetitions of the same class are allowed), for transitions it is expressed as a pair $\langle$ variable types, guard $\rangle$ defining the possible values that can be assigned to transition variables in a transition instance; guards must be expressed in the form of* standard predicates,
4. **Pre**$[p,t]$, **Post**$[p,t] : cd(t) \rightarrow \mathrm{Bag}(cd(p))$ *are the* pre- *and* post- incidence matrices, *expressed in the form of* arc functions,
5. $m_0 : m_0(p) \in \mathrm{Bag}(cd(p))$ *is the initial marking of place p.*

2.2 SRG

The syntax of Well-formed Nets makes it possible to directly build a quotient graph, the so-called Symbolic Reachability Graph (SRG). Classes group markings that have the same distribution of tokens in places but differ only by the identity of tokens. Hence, a class can be built by forgetting the colors of tokens and replace them by variables. The markings represented by the class correspond to the possible instantiations of the variables.

Of course, a variable is associated with a Cartesian product of object classes and can be instantiated only by colors belonging to that product. However, not any instantiation is allowed within a class. If we want the SRG to preserve the firing sequences of the original RG, all the markings belonging to the same class must correspond to states in which the system behaves similarly.

In the WN formalism, the similarity of behavior is taken into account by the definition of static subclasses. If we consider a marking *m* obtained by some instantiation, only the markings obtained from *m* by a permutation of objects that respects static subclasses belong to the same class as *m*.

As a consequence, the cardinality of the classes of markings, hence the efficiency of the approach, is directly related to the cardinality of static subclasses. This is all the more true if there are ordered classes in the model. Actually, only rotations are allowed among objects of an ordered class thus reducing the possibility of grouping markings. The worst case appears when there are static subclasses inside an ordered class. In this case, an object of the class can never be substituted another object of the same class in an attempt to find another marking belonging to the same class. It is thus crucial that static subclasses be defined in a maximal way, i.e., that objects with similar behaviors do not belong to different static subclasses.

3 Symmetry of a model and representation

The goal of this work is to show that the partition into static sub-classes can be automatically deduced from the model itself. A partition of color domains into static sub-classes is such that an unfolding of the colored net into a black and white one will produce

the same motif for any two elements of a given sub-class. This means that static sub-classes capture the **structural** symmetries of the model. As each element of a colored net is liable to introduce structural asymmetry, our goal is to independently analyze all the components of the model, to deduce what structural asymmetry -if any- it induces.

We start our exploration of the model with the basic assumption that all classes are symmetric and unordered. Of course for a net without any places or transitions this assumption is true. As each element is analyzed, the asymmetry it induces is added (by refinement) to the partitioning of the classes corresponding to the global structural symmetry.

Definition 4. Partition refinement *Let C be a color class and Π_1, Π_2 be two partitions of C. We recursively define the refinement $\Pi = \Pi_1 \sqcap \Pi_2$, as a partition of C obtained by intersecting every set S_1 in Π_1 with every set S_2 in Π_2:*

- *if $S1 = S2$ a set $S = S1$ is added to Π, the construction proceeds with the next $S1$ from Π_1*
- *if $S1 \cap S2 = \emptyset$ the construction proceeds, no sets are added to Π.*
- *if $S1 \cap S2 \neq \emptyset$ the set $S1 \cap S2$ is added to Π, and the construction proceeds with $S_1 \backslash S_2$ in lieu of S_1.*

Example 2. For instance, let $C = \{1,2,..6\}$, $\Pi_1 = \{1,2,3\} \uplus \{4,5,6\}$ and $\Pi_2 = \{1,2\} \uplus \{3,4\} \uplus \{5,6\}$, $\Pi = \Pi_1 \sqcap \Pi_2 = \{1,2\} \uplus \{3\} \uplus \{4\} \uplus \{5,6\}$.

A partition of a color domain is a way to represent allowable symmetries : it represents all permutations that preserve the partition, i.e. any bijective function $f : C \to C$ such that $\forall e \in C_i \subset \Pi$, $f(e) \in C_i$.

Thus we start by considering that all elements of a color class C are interchangeable by defining a partition composed of a single static sub-class containing all its elements. As we find elements of the net that break the symmetry, we extract the partition corresponding to the symmetry they allow, and obtain the new globally admissible symmetries by common refinement of these two partitions. This process is iterated until either all the elements of the considered color class have been separated into an individual subclass, or all the components of the net have been analyzed.

A partition of a class of n elements into n sub-classes is a worst-case scenario, in which symbolic study will yield the same results as the classic approach. It corresponds to a totally asymmetric color domain.

An important aspect of the symmetry of a color class is whether it is ordered or not. In *WN* an ordered class is necessarily circular, and allows use of the successor operator. On ordered classes the only permitted permutations are rotations, thus the circularity helps to increase the symmetry of a problem.

For instance in the classic philosophers problem, we wish to distinguish the relative positions of the philosophers around the table (i.e. $!P$ is left of P), but not set a "first" philosopher. In this manner, we can study what happens when **a** philosopher has started eating, and his left neighbor wishes to eat, instead of studying independently *Phi*0 is eating and *Phi*1 wishes to eat, *Phi*1 is eating and *Phi*2 wishes to eat ... etc.

An exploration of the components of the net is performed to find any use of the successor operator on a given class, thus giving it the status "ordered". A problem sometimes encountered is when we have an ordered class from which we wish to distinguish

an element or group of elements. Since the element thus distinguished is structurally different from the others of the class, its predecessor has to be distinguished, since the successor operation when applied to it yields a structurally different token. Recursively, we obtain that we have to distinguish all the elements from each other, thus falling in our worst-case scenario described above.

4 Components of net and induced (a)symmetry

By taking advantage of symmetries, *WN* offer the possibility of efficient model-checking, but the syntax is a bit constraining. It requires careful study to maximize the potential for symbolic analysis of the reachability graph.

The model being self-contained, the actual (a)symmetries are defined by the model itself. The goal of our work is to identify and exploit the information that induces asymmetry in the model *automatically*, thus allowing to use *WN* with a less constrained syntax. This section successively studies the effect of place markings, arc color functions and transition guard predicates on the model symmetries.

4.1 Places

First let us consider a place P of domain $cd(P)$ composed of a single class $C = cd(P)$: a marking in a *WN* is a function from $\mathcal{M} : P \rightarrow \text{Bag } (cd(P))$, and is noted $\mathcal{M}(P) = \sum \alpha_i C_i.ALL$, $\alpha_i \in \mathbb{N}$, where C_i are static sub-classes of C. Let $C = \{e_1, \ldots, e_n\}$ be its elements. We allow markings to directly reference these elements, therefore markings have the form $\mathcal{M}' : P \rightarrow \text{Bag } (\{e_1, \ldots, e_n\})$. In other words we allow markings to directly reference the elements of a color domain. This form can easily be rewritten in the former using the following technique: let $\Pi = \Pi_0 \cup \ldots \cup \Pi_k$ be a partition of C such that the elements of Π_i are present exactly i times in in place P. This partition is trivially the minimal partition into static sub-classes that allows to describe $\mathcal{M}(P)$, and describes the asymmetry generated by the marking of this place. The marking of P is therefore expressed as $\mathcal{M}(P) = \sum_i i \cdot \Pi_i.ALL$.

Example 3. For example, let C be a class composed of $\{e1..e6\}$ and the marking of a place P be

$\mathcal{M}(P) = \langle e1 \rangle + \langle e1 \rangle + \langle e2 \rangle + 2 * \langle e3 \rangle + \langle e4 \rangle$,

We rewrite this under the form:

$\mathcal{M}(P) = 0 * (\langle e5 \rangle + \langle e6 \rangle) + 1 * (\langle e2 \rangle + \langle e4 \rangle) + 2 * (\langle e1 \rangle + \langle e3 \rangle)$,

In this form we obtain:

$\Pi_0 = \{e5, e6\}; \Pi_1 = \{e2, e4\}; \Pi_2 = \{e1, e3\}$.

This partition of C defines the asymmetry induced by the place P. We represent such a marking under the form:

$\mathcal{M}(P) = \langle [e2, e4] \rangle + 2\langle [e1, e3] \rangle$.

However in the case of a composite color domain the calculation is more difficult. We use an intermediate notation to obtain a homogeneous form for individual marks: in *WN* syntax, a marking is of the form $\mathcal{M}(P) = \sum \alpha_i L_i, \alpha_i \in \mathbb{N}$, and L_i are tuples of the form $\langle C_{i1,j1}.ALL, .., C_{i_n,j_n}.ALL \rangle$ where a $C_{i,j}$ represents the j^{th} static subclass of color class C_i.

Example 4. For example $\mathcal{M} = 2\langle C_0, D_1\rangle + \langle C_1, D_0\rangle$ is a possible marking $\mathcal{M}$ for a place of color domain $C \times D$, if we do not represent the *.ALL* to simplify the notation. Let us now assume that $C = C_0 \uplus C_1, C_0 = \{a,b\}, C_1 = \{c\}$ and $D = D_0 \uplus D_1, D_0 = \{x\}, D_1 = \{y,z\}$, this marking represents the following concrete marking: $\mathcal{M} = 2*(\langle a,y\rangle + \langle a,z\rangle + \langle b,y\rangle + \langle b,z\rangle) + \langle c,x\rangle$. Since the subclasses are not yet defined when we begin our analysis, we use a notation explicitly enumerating the elements of a subclass. For example we write:

$\mathcal{M} = 2\langle [a,b],[y,z]\rangle + \langle [c],[x]\rangle = 2(\langle [a],[y,z]\rangle + \langle [b],[y,z]\rangle) + \langle [c],[x]\rangle$

$\mathcal{M} = 2*(\langle [a],[y]\rangle + \langle [a],[z]\rangle + \langle [b],[y]\rangle + \langle [b],[z]\rangle) + \langle [c],[x]\rangle$

We turn up here upon the beginnings of some algebraic laws that might allow us to factorize a given concrete marking into a reduced symbolic one. Let a vector of elements L_i represent the tuple of a token, and a multiplicity α_i assigned to a token represent a *mark M*, and a list or sum of marks represent a marking $\mathcal{M}$. There are generally different equivalent *representations* or formulae F of a given marking $\mathcal{M}$, as shown in example 4 that exhibits three representations of a marking $\mathcal{M}$. Let us define more precisely the process of factorization:

Definition 5. Marks:

A mark is of the form $M = \alpha_i L_i$, $\alpha_i \in \mathbb{N}, L_i = \langle v_1,..,v_n\rangle$ *with* v_i *a vector of elements of a color domain.* α_i *is the multiplicity of the mark in the considered marking and* L_i *is its associated tuple. Let* $M = \alpha L, L = \langle v_1,..,v_n\rangle$, *and* $M' = \alpha' L', L' = \langle v'_1,..,v'_n\rangle$. *The following rules hold true:*

Marking Factorization:

Let F_1 *and* F_2 *be representations of a marking* $\mathcal{M} = F_1 = F_2$. F_1 *is more factorized than* F_2 *if less marks are used to represent it. Since for any non-empty marking at least one mark is necessary, we define the factorized form of a marking as any form that uses a minimal number of marks to represent it.*

Multiplicity distributivity rule:

$\forall \alpha$ *and* $\alpha', \alpha L + \alpha' L = (\alpha + \alpha')L$

Merge rule:

At equal multiplicity, if L *and* L' *have a single mismatch* $m \neq m'$, *such that* $L = \langle v_1,..v_{k-1}, m, v_{k+1},..v_n\rangle$ *and* $L' = \langle v_1,..v_{k-1}, m', v_{k+1},..v_n\rangle$, $M + M'$ *can be merged into a single symbolic marking, by defining an additive operation on vectors of elements as the union operator on sets:*

$\alpha = \alpha', L = \langle v_1,..v_{k-1}, m, v_{k+1},..v_n\rangle$ *and* $L' = \langle v_1,..v_{k-1}, m', v_{k+1},..v_n\rangle$

$\Leftrightarrow M + M' = \alpha\langle v_1,..v_{k-1}, m \cup m' \setminus m \cap m', v_{k+1},..v_n\rangle + 2\alpha\langle v_1,..v_{k-1}, m \cap m', v_{k+1},..v_n\rangle$.

To simplify our calculations we will depend upon the underlying data structure to ensure that $m \cap m' = \emptyset$. This property is easily obtained if the algorithm starts from a completely developed form (i.e $\forall v_i, |v_i| = 1$), and the multiplicity distributivity rule is systematically applied before any merging occurs. Example 5 exhibits an application of these rules.

Lemma 1. *Even at equal multiplicity, if* L *and* L' *have more than one mismatch, no factorization of* $M + M'$ *is possible.*

Proof. The number of concrete marks represented by a symbolic mark is obtained by multiplying the cardinality of the v_i. We initiate a recursion by considering a tuple composed of two color domains. Let $M'' = M + M' \Leftrightarrow \langle v''_1, v''_2 \rangle = \langle v_1, v_2 \rangle + \langle v'_1, v'_2 \rangle$, we obtain $|M''| = |v''_1| * |v''_2| = |M| + |M'| = |v_1| * |v_2| + |v'_1| * |v'_2|$. Since all the individual elements mentioned in v_1 and v'_1 [resp.v_2 and v'_2] must be mentioned in v''_1 [resp. v''_2], we obtain that $v''_1 = v_1 \uplus v'_1$ and $v''_2 = v_2 = v'_2$ (or vice versa). Because a cardinality is a strictly positive integer, $|M''| = |M| + |M'|$ is true if and only if $|v''_1| = |v_1| + |v'_1|$ and $|v''_2| = |v_2| = |v'_2|$ or vice versa. In other words there is a single mismatch between M and M' or M'' cannot be defined. This result is trivially extensible to tuples of arbitrary length.

Corollary 1. *Any form for a marking $\mathcal{M}$, in which any two marks have two or more mismatch is in factorized form.*

An algorithm computing such a form tries to match off pairs of marks, if their tuples L are equal it applies the multiplicity rule, if they have one mismatch it applies the merge rule, and if they have more than a single mismatch it tries the next pair. Because when a new mark is produced by applying the merge rule, it has to be compared (again) to all the other marks we obtain a combinatorial effect. The total complexity of the algorithm is in $O(setcard * tuplesize * nbterm!)$, where *setcard* is the cardinality of a mentioned set, *tuplesize* is the number of elements that constitute a tuple, and *nbterm* is the number of terms encountered in the marking expression. This complexity is therefore strongly dominated by the factorial effect of the number of terms in a given formula, and the size of the domains studied only feebly impacts the complexity through *setcard*. This complexity is manageable, as most marking functions contain a limited number of terms. The total asymmetry generated by a given marking is thus calculated by intersecting the sets mentioned in each mark of the factorized marking. Once this partition has been found, we trivially rewrite the marks in term of this partition, because for any v_i of a mark of the factorized marking, $v_i = C_{j_1} \uplus .. \uplus C_{j_n}$ where $C = C_1 \uplus .. \uplus C_n$ is the partition obtained for color domain C that v_i is part of. In other words the partition into static subclasses is "finer grain" than that encountered in any set mentioned.

Example 5. Let us study the asymmetry generated by the marking:

$\mathcal{M} = 2 * (\langle [a],[y] \rangle + \langle [a],[z] \rangle + \langle [b],[y] \rangle) + \langle [b],[z] \rangle + \langle [b],[z] \rangle + \langle [c],[z] \rangle$

By applying distributivity we obtain:

$\mathcal{M} = 2 * (\langle [a],[y] \rangle + \langle [a],[z] \rangle + \langle [b],[y] \rangle + \langle [b],[z] \rangle) + \langle [c],[z] \rangle$

Then at equal multiplicity, $\langle [a],[y] \rangle + \langle [a],[z] \rangle \Rightarrow$ Single mismatch $\Rightarrow \langle [a],[y,z] \rangle \ldots$

$\mathcal{M} = 2\langle [a,b],[y,z] \rangle + \langle [c],[z] \rangle$ which in turn gives us $C = (C_0 = [a,b]) \uplus (C_1 = [c])$ and $D = (D_0 = [y]) \uplus (D_1 = [z]) \uplus (D_2 = [x])$ since the $\langle c,z \rangle$ term forces to isolate z from y.

Thus we obtain $\mathcal{M} = 2(\langle [a,b],[y] \rangle + \langle [a,b],[z] \rangle) + \langle c,z \rangle$, and $\mathcal{M} = 2(\langle C_0, D_0 \rangle + \langle C_0, D_1 \rangle) + \langle C_1, D_1 \rangle$ where *.ALL* is implicit. Since all the marks mentioned now have at least two mismatches, this symbolic marking is the factorized minimal representation of the initial concrete marking in *WN* syntax.

4.2 Arcs

WN basic arc functions are limited to diffusion-synchronization over a static sub-class, identity and successor functions.

- Identity is a totally symmetric function and thus does not introduce asymmetry in the model
- The successor function implies an ordering of the associated color domain. The first occurrence of a successor function on a variable X of a color domain C will add the tag ordered to this class. Any subsequent partitioning of C into sub-classes will give rise to a worst-case scenario in which each element of color C will be isolated in a static sub-class containing only that element. A class thus partitioned will be ignored in further analysis, as no further gain for symbolic exploration is possible.
- The diffusion-synchronization function separates the sub-classes it mentions from the rest of the elements of the class. Thus the static sub-classes mentioned are part of the minimal partition expressing the asymmetry generated by the considered arc, and will be used to further refine the partition corresponding to the global asymmetries.

In addition to these "classic *WN*" color functions, we add the possibility to directly reference elements e_i of class C:

- The constant reference function expressed as $\sum \alpha_i e_i$ (i.e. direct reference to constant elements e_i of C with multiplicity α_i) can be rewritten in terms of the diffusion-synchronization function given an appropriate partition Π of C of the same form as that defined for markings. Thus, though this is an extension of syntax for ease of use, the semantics of *WN* are preserved.

The asymmetry generated by a color function is computed like the asymmetry generated by a marking. The only difference is that a color function may reference formal parameters instead of just sets (our v_i in the study of the marking).

Definition 6. Color function:
A color function is defined as $\mathcal{F} = \sum \alpha_i L_i$, *with*

- $L_i = \langle V_1, ..V_n \rangle$ *and*
  - $V_i = [e_{i_1}, ..e_{i_n}]$ *a vector of elements representing a subclass, or*
  - $V_i = [(!)var_1, .., (!)var_n]$ *a list of formal parameters with possible use of the successor operator.*

Mismatch *A vector of elements always mismatches a list of variables, and they cannot be added. A variable mismatches any other variable, including itself with a successor operator. Lists of formal parameters can be added using the union operation on sets.*

The rules defined on markings in the previous section apply with this enhancement of the comparison operation.

Example 6. Let $\mathcal{F} = \langle [!X], [a] \rangle + \langle [!X], [b] \rangle + \langle [X], [a,b] \rangle$ be a color function, where X is a formal parameter and $a, b \in C$ a color domain.
$\langle [!X], [a] \rangle + \langle [!X], [b] \rangle \Rightarrow$ Single mismatch $\Rightarrow \langle [!X], [a,b] \rangle$
$\langle [!X], [a,b] \rangle + \langle [X], [a,b] \rangle \Rightarrow$ Single mismatch $\Rightarrow \langle [X, !X], [a,b] \rangle$
This defines $C_0 = [a,b]$ as a candidate static subclass of C

Though defining the possibility of mentioning several variables in a single V_i allows for more compact representation, it does not put in evidence additional symmetry of the model. Since only the actual element sets mentioned influence the computation of the partition of C into subclasses, and that merging on a variable implies that all the V_i match, the partition obtained under any form is the same.

4.3 Transitions

The study of transitions and guard predicates is slightly more complex than that of markings or color functions, and is the focus of the next section. This is due to the fact that we wish to allow expressions that are Boolean functions (AND $\wedge$,OR $\vee$ and NOT $\neg$) of basic predicates, instead of the basic additive composition allowed in both marking and color function definition. We will present here the basic predicates that we wish to allow, and show how we can write them in a homogeneous normalized way.

Definition 7. Normalized Basic Predicate*: A normalized basic predicate is a domain restriction predicate of the form $X \in E$, where X is a variable and E is a set of elements of the color domain C of X, or is a variable comparison predicate of the form $X = Y$ or $X \neq Y$ possibly with use of the successor* ! *operator. The set E of a domain restriction predicate will be referred to as the acceptation set of this predicate.*

We allow the "classic" *WN* basic predicates:

- $X = Y$ **synchronization**: this predicate and its dual $X \neq Y$ is permitted as is, since they do not introduce additional asymmetry. More precisely, its effect on the model is already correctly taken into account when building the SRG.
- $X =!Y$ **successor**: this predicate implies that the class X and Y belong to is ordered. Thus it will be treated in the same way as the successor color function found on arcs, by adding the information that class C is ordered. It should be noted again that this information is extremely constraining, and modifies the way subsequent partitions of C are treated. As a consequence, a first exploration of the model will be performed looking for these successor functions before beginning any analysis, to reveal which classes should be considered ordered.
- $d(X) = d(Y)$ **domain synchronization**: this predicate introduces all the static sub-classes as initially introduced by the modeler. At most, the partition Π corresponding to the asymmetry introduced by this predicate -if by itself- is the partionning of C into its static sub-classes C_i. This type of predicate is normalized as $\bigvee_i((X \in C_i) \wedge (Y \in C_i))$.
- $d(X) = C_i$ **domain identification**: This predicate introduces the sub-class C_i, and is written $(X \in C_i)$ in our normalized form

In addition to the "classic *WN*" predicates, we introduce the possibility of explicitly referencing elements of a color domain:

- $X = e_i$ and $X \neq e_i$ **constant reference**: allows to directly compare a variable X to an element of its color domain. This predicate will be normalized in the form $(X \in \{e_i\})$ (respectively $(X \in C \backslash e_i)$)

– $X \in E$ **explicit partition**: where E is a set explicitly naming the referenced elements (i.e.: $state \in \{interrupted, error\}$). This is the basic predicate in our normalized form.

For completeness reasons, and to provide shorthands, we will also allow use of comparison operators. It should be noted that these operators are forbidden on *WN* because if a class is unordered the meaning is unclear, and if it is ordered it is circular therefore the assertion is even more meaningless. Despite this, there is a commonly understood way of interpreting these assertions, if considering classes to be non-circular ; the ordering that is referred to is given by the definition of the color, but is independent and incompatible with the ordering as defined by the successor operator:

– $X < e_i$ **constant comparison**: This will be interpreted as a shorthand referring to the $i-1$ first elements of the class, the ordering being given by the definition of the color domain. It is thus normalized in the form $X \in \{e_1, .., e_{i-1}\}$. We will similarly allow use of $\leq, >, \geq$.
– $X < Y$ **variable comparison**: (X and Y variables of same color domain) In the same way, this predicate can only have meaning if the class is considered non-circular. This predicate gives rise to a worse case scenario for the color C concerned, as we will have to explicit all the possible cases in the normalized form: $\bigvee_{i=1}^{n}((X \in \{e_1, .., e_i\}) \wedge (Y \in \{e_{i+1}, .., e_n\}))$. We will similarly allow use of $\leq, >, \geq$.

However, we emphasize that use of these operator does **not** introduce an ordering on the class in the sense of a successor operation, it even forbids use of successor in the rest of the net.

5 Study of guard predicates

Because the study of guard predicates is more complex than that of markings or color functions it is separated in this section. The technique developped in this section relies on a canonization process, such that the canonized form we obtain directly exhibits the symmetries allowed by a guard. We rely on a binary tree representation of the Boolean formula of the guard to ensure a unicity property essential to the algorithm (5.2). It requires a separate application of the canonization process for each formal parameter mentionned in the guard (5.3). We further prove that the symmetries exhibited by this canonized form are **strictly** the symmetries allowed by the guard (5.4), and outline an algorithm to obtain it (5.5).

5.1 Foreword: Normalized forms of predicates

As we have seen, the syntax proposed allows complex expressions for guard. To clarify the goal of this section, we will use a small example that exhibits the problems encountered and solved by our approach.

Example 7. Let us consider two color domains $C = \{1, 2, 3, 4, 5\}$ and $D = \{a, b, c\}$, and two variables $X \in C$ and $Y \in D$. If we consider a guard G where $\vee$ denotes OR and $\wedge$ denotes AND:

$G = (X \leq 3 \wedge Y = a) \vee (X \in \{3,4,5\} \wedge Y > a)$

With the normalized form described above we obtain:

$G = (X \in \{1,2,3\} \wedge Y \in \{a\}) \vee (X \in \{3,4,5\} \wedge Y \in \{b,c\})$

Our algorithm will allow us to obtain the form (normalized over X):

$G = (X \in \{1,2\} \wedge Y \in \{a\}) \vee (X \in \{3\} \wedge Y \in \{a,b,c\}) \vee (X \in \{4,5\} \wedge Y \in \{b,c\})$

Giving the partition $C = (C_0 = \{1,2\}) \uplus (C_1 = \{3\}) \uplus (C_2 = \{4,5\})$. Then by normalizing over Y we obtain:

$G = (Y \in \{a\} \wedge X \in \{1,2,3\}) \vee (Y \in \{b,c\} \wedge X \in \{3,4,5\})$

Giving the partition $D = (D_0 = \{a\}) \uplus (D_1 = \{b,c\})$. The final form expressed in terms of subclasses would be:

$G = (X \in C_0 \wedge Y \in D_0) \vee (X \in C_1 \wedge (Y \in D_0 \vee Y \in D_1)) \vee (X \in C_2 \wedge Y \in D_1)$

We will explain here how these different forms are computed, and prove that the obtained expression minimally represents the asymmetry induced by a guard. The rest of this section is more involved and the casual reader should skip to the *Implementation and results* in section 6.

5.2 Predicate data representation

We use a canonical binary tree for a guard G's representation:

- the leaves of the tree, noted L_i, are normalized predicates of the form $(X \in P)$ as introduced in section 4.3
- the internal nodes are the operators AND $\wedge$ and OR $\vee$.

The negation is directly applied to obtain the dual form of a negated sub-expression (i.e. $\neg(X \in E)$ becomes $(X \in C \backslash E)$). Variable comparison predicates are switched as needed to obtain a lexicographical order (i.e. $X = Y$ and not $Y = X$). This makes comparison of subtrees easier.

To apply the normalization algorithms, we require an essential unicity constraint, that will ensure that any two sets mentioned in predicates L_i over the same variable have an empty intersection. Let L_1 and L_2 be two leaf predicates of a guard G:

Property 1. **Unicity:**
$\forall L_1, L_2 \in G$ such that $L_1 = (X \in P_1)$ and $L_2 = (X \in P_2)$,
$P_1 \cap P_2 = \emptyset$
or $P_1 = P_2$ and L_1 and L_2 share a single physical representation.

Our tree structure uses the following algorithm to ensure unicity of representation by construction.

Algorithm ensuring unicity
When inserting a leaf predicate $L = X \in E$ in a guard structure G:

- Explore G to find any predicates of the form $L_i = X \in E_i$ already present in G
- Compute the intersection $E \cap E_i$ of the new predicate set E with previously constructed ones E_i, for all i:
  - $E = E_i$: interrupt the iteration by returning the physical address of the $X \in E_i$ leaf ; *L and L_i will share a single physical representation.*

- $E \cap E_i = \emptyset$: no action is taken, iterate for the next E_{i+1} set ; $P \cap P_i = \emptyset$
- $E \cap E_i \neq \emptyset$: Compute a partition of $E \cup E_i$ such that $E \cup E_i = P_1 \uplus P_2 \uplus P_3$, such that $P_1 = E \backslash E_i$, $P_2 = E \cap E_i$ and $P_3 = E_i \backslash E$. Then create three leaves L_1, L_2, and L_3 corresponding to these sets (i.e. $L_1 = (X \in P_1)$ etc.). Finally modify **in place** the leaf node $L_i = (X \in E_i)$ into an OR $\vee$ node of left son L_2 and right son L_3. The construction ensures that these sets (P_2 and P_3) do not intersect any other already present in G. Continue iteration but using L_1 instead of L ; P_1, P_2 *and* P_3 *do not overlap.*

– When this iteration is concluded, we construct a tree composed of an OR between the L_i and the resultant L that constitute the partition of the original E (i.e. the P_1 and P_2 of the iteration phase), and return the physical address of its root.

Insertion of a TRUE or FALSE statement, or of a variable comparison leaf follows a similar procedure in that only one physical instance of any of these will ever be created for a given tree G.

Inserting an internal operator node $A\ op\ B$ uses the above procedure to obtain A and B (recursively) then creates an op node of sons A and B and returns it.

Thus this data structure, by **construction** ensures the unicity property.

5.3 Normal developed form w.r.t. a formal parameter

The goal here is to define a normal form for the guard expression G considered that allows to calculate the asymmetry induced by G. This normal form will be computed independently for each formal parameter (variable) mentioned in G, as shown in the foreword (Example 7).

Definition 8. Normal developed form

Let the developed form of a guard G with respect to a formal parameter X be:
$G = (\sum_i X_i \wedge M_i) \vee (\sum_j M_j)$

Where X_i is a predicate over variable x of the form $(x \in E_i)$, such that for all $i,k, i \neq k$, $E_i \cap E_k = \emptyset$, and M_i, M_j are Boolean guard expressions that do not mention variable x except possibly through a variable comparison, with the additional constraint that no two M_i be equal.

Property 2. **Normal Form property** The permutation of an instance e_1 of formal parameter x with an instance e_2 does not modify the evaluation of the guard **if and only if** e_1 and e_2 belong to the acceptation set of the same predicate in the developed form of the guard formula.

The proof of this property may be found at the end of the next section, as it requires some further definitions.

Thus the developed form defines a **minimal** partition of a color domain C, for a given variable x, as the partition of C into the acceptation sets of the X_i predicates of the developed form. The tokens in a given E_i are structurally equivalent with respect to the considered guard, any permutation of elements that respects this partition leaves the truth value of the guard predicate unchanged.

This is the property that defines static sub-classes of a *WN*, thus it defines the asymmetry induced by the guard over the color domain C of the formal parameter considered. To obtain the full set of asymmetry induced by a guard, we need to compute the canonized form for all formal parameters mentioned, and refine the partitions obtained if more than one parameter belongs to the same color domain.

5.4 Computation of the normal form

Now that we have defined the form we wish to obtain to explicit the symmetry allowed by a guard, we need an algorithm to obtain it. To this end we define the *depth* of a node in an expression tree, such that **a predicate on a variable X is under developed form if its depth is at most 1**:

Definition 9. Node depth

Let the depth of a node be recursively defined as follows:

- *We define the depth of the root's parent node as* **0**.
- *Let n be the depth of the parent of the node studied.*
  - *if* $n = 0$*:*
    - *If the node is a leaf or a* $\vee$ *OR operator its depth is* 0
    - *Else the node is a* $\wedge$ *AND operator and its depth is* 1
  - *else* $n > 0$*:*
    - *If the node is a leaf its depth is n*
    - *Else the node is an operator and its depth is* $n+1$

To reduce the depth of predicates on the target variable, we apply the classic laws of Boolean algebra (table 1), where 0 represent *FALSE* and 1 represents *TRUE*.

Name	Expression	Dual
Identity:	$x \vee 0 = x$	$x \wedge 1 = x$
Commutativity:	$x \vee y = y \vee x$	$x \wedge y = y \wedge x$
Distributivity:	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
Complementarity:	$x \vee \neg x = 1$	$x \wedge \neg x = 0$
Idempotency:	$x \vee x = x$	$x \wedge x = x$
Associativity:	$x \vee (y \vee z) = (x \vee y) \vee z = x \vee y \vee z$	$x \wedge (y \wedge z) = (x \wedge y) \wedge z = x \wedge y \wedge z$
Absorption:	$x \vee (x \wedge y) = x$	$x \wedge (x \vee y) = x$
Morgan's law:	$\neg(x \vee y) = \neg x \wedge \neg y$	$\neg(x \wedge y) = \neg x \vee \neg y$
Dominance:	$x \vee 1 = 1$	$x \wedge 0 = 0$
Involution:	$\neg(\neg x) = x$	

Table 1. Boolean Algebra laws

We add the following operation, derived from the form for of basic predicates, that is similar to our merge rule defined on markings:

$$\begin{aligned} \mathit{Predicate\ fusion}: (X \in E_1) \vee (X \in E_2) &= (X \in E_1 \cup E_2) \\ \mathit{Dual}\ (X \in E_1) \wedge (X \in E_2) &= (X \in E_1 \cap E_2) \end{aligned}$$

We further define an operation essential to the factorization process, and that is made possible by the unicity property (1):

Definition 10. Quotient tree:

Let X be a leaf predicate bearing on variable x of the form $(x \in E)$, and T be an expression tree, the **quotient tree** *T' of T by X, noted $T' = T/X$ is obtained by replacing any leaf predicate of T of the form $(x \in F)$ bearing on variable x by $TRUE$ if $E = F$, and $FALSE$ if $E \neq F$.*

This definition ensures the following property:

Property 3. A quotient tree $T' = T/X$ does not contain any leaf predicate bearing on variable x, and $T \wedge X = T' \wedge X$.

This definition and subsequent property is the result of applying distributivity and dual predicate fusion rules.

Proof. Let $F = T \wedge X$ and $F' = T' \wedge X$,

- If the assertion X is false, both F and F' are false (Dominance)
- Else if X is true, the evaluation of T will evaluate any predicate on x as true if it is X and false if it is a different predicate, because the acceptation sets are distinct (unicity property). This is precisely the way T' is defined.

We can now prove the normal form property given above:

Proof. **Normal form:**(proof of property 2)

Let us remind that a normal form w.r.t. a formal parameter x is defined as:
$G = (\sum_i X_i \wedge M_i) \vee (\sum_j M_j)$

Where the X_i are predicates over variable x of the normalized form $(x \in E_i)$, such that the acceptation sets of any two X_i be distinct, and M_i, M_j are Boolean guard expressions that do not mention variable x except possibly through a variable comparison, with the additional constraint that no two M_i be equivalent.

Let $(F)_{x=e_i}$ be the value of formula F when evaluated with the constraint $x = e_i$, e_i being a constant value of the color domain C of variable x. The following equivalences hold:

$$
\begin{aligned}
(F)_{x=e_1} = (F)_{x=e_2} &\Leftrightarrow ((x \in \{e_1\}) \wedge F)_{x=e_1} = ((x \in \{e_2\}) \wedge F)_{x=e_2} \text{(Identity dual)} \\
&\quad \text{Let } F'_1 = F/(x \in \{e_1\}) \text{and } F'_2 = F/(x \in \{e_2\}). \text{Thus} \\
&\Leftrightarrow ((x \in \{e_1\}) \wedge F'_1)_{x=e_1} = ((x \in \{e_2\}) \wedge F'_2)_{x=e_2} \\
&\quad \text{Since } (x \in \{e_1\})_{x=e_1} = (x \in \{e_2\})_{x=e_2} = TRUE \\
&\quad \text{And since a quotient tree does not mention variable } x, \\
&\quad (F'_1)_{x=e_1} = F'_1 \text{ and } (F'_2)_{x=e_2} = F'_2, \\
(F)_{x=e_1} = (F)_{x=e_2} &\Leftrightarrow F'_1 = F'_2
\end{aligned}
$$

F is of the form $F = (\sum_i X_i \wedge M_i) \vee (\sum_j M_j)$ Let $i1$ and $i2$ be such that $X_{i1} = (x \in E_{i1})$ and $X_{i2} = (x \in E_{i2})$ be two predicates such that $e_1 \in E_{i1}$ and $e_2 \in E_{i2}$.

Thus, $F'_1 = M_{i1} \vee \sum_j M_j$ and $F'_2 = M_{i2} \vee \sum_j M_j$. Therefore:

$$\begin{aligned}
(F)_{x=e_1} = (F)_{x=e_2} &\Leftrightarrow F'_1 = F'_2 \\
&\Leftrightarrow M_{i1} = M_{i2} \\
&\Leftrightarrow i1 = i2 \text{ since the } M_k \text{ are all distinct} \\
&\Leftrightarrow \exists i, \text{suchthat } X_i = x \in E_i \text{and that } e_1 \in E_i \textit{ and } e_2 \in E_i \\
(F)_{x=e_1} = (F)_{x=e_2} &\Leftrightarrow e_1 \text{ and } e_2 \text{ belong to the acceptation set of} \\
&\qquad \text{the same term of } F \text{ in developped form}
\end{aligned}$$

Thus the permutation of an instance e_1 of formal parameter x with an instance e_2 does not modify the evaluation of the guard **if and only if** e_1 and e_2 belong to the acceptation set of the same predicate in the developed form of the guard formula.

5.5 Algorithm to obtain a developed form

The full algorithm used to obtain a normal form is composed of four phases:

- we first *develop* the terms on the target variable x to have them all at depth 0 or 1, thus obtaining a sum of products form.
- we then *merge* the identical occurrences of predicates on x to obtain a single occurrence of each by comparing them.
- we then *canonize* the M_i terms to allow their comparison (we need to obtain $\forall i, j$ $M_i = M_j \Leftrightarrow i = j$)
- Finally we *group* X_i of same M_i.

Example 8. Let $\mathcal{G} = (M_2 \vee X_1) \wedge (M_1 \wedge X_2) \vee X_1 \wedge M_3$, where $X_1 = x \in E_1, X_2 = x \in E_2$:

- Phase 1: $\mathcal{G} = X_1 \wedge M_1 \vee M_1 \wedge M_2 \vee X_2 \wedge M_2 \vee X_1 \wedge M_3$. The $X_1 \wedge X_2$ necessarily evaluates to $false$ because of the unicity property.
- Phase 2: $\mathcal{G} = X_1 \wedge (M_1 \vee M_3) \vee X_2 \wedge M_2 \vee M_1 \wedge M_2$
- Phase 3: Let us suppose that canonization has shown $M_1 \vee M_3 = M_2$.
- Phase 4: $\mathcal{G} = (X_1 \vee X_2) \wedge M_2 \vee M_1 \wedge M_2 = (x \in E_1 \cup E_2) \wedge M_2 \vee M_1 \wedge M_2$ which is the normalized form for $\mathcal{G}$ over x, and defines $E_1 \cup E_2$ as a candidate subclass.

For lack of space we will not develop the four aforementioned phases of the algorithm, but they may be obtained by contacting the authors.

6 Implementation and results

To illustrate our work we consider a part of an industrial application, an Electrical Flight Control System [7], that manages the spoilers of an airplane during takeoff and landing. Opening angles applied to the spoilers are computed with respect to the value given by sensors (altitude, speed, ...) and the angle the pilot wants to apply.

We present the function that allows to detect if the airplane is on ground by checking the altitude, speed of the wheels and weight on the wheels. The altitude and speed

sensors are monitored and a corrupt behavior leading to false information may be detected. The possible values provided by the altitude sensor are in the set composed of $\{Altitude\_Min..Altitude\_Max\} \cup \{Altitude\_Corrupt\}$. In the same way, the speed sensor may return any value in $\{Speed\_Min..Speed\_Max\} \cup \{Speed\_Corrupt\}$. We represent each sensor domain by a place that initially contains all the possible values, therefore considering all possible sensor values in our model-checking.

The main difficulty when formally studying such a application is the infinite domain of possible values produced by the continuous domain sensors. A first step towards analysis is to represent these infinite domains by discrete ones. Even after this operation we obtain domains too large for conventional state graph exploration. Therefore, it is necessary to automatically detect the limit values of speed and altitude that have a relevant impact on the behavior of the model.

The Petri net model of the studied function, with classes and variables declaration, is represented by Fig.1. Places *Pi* correspond to the successive steps of the computation of the function result. Place *WeightPossibleVal* contains all the possible values for a sensor weight. Places$Weight_Left_Wheel$ and $Weight\_Right\_Wheel$ represent the value produced by the sensor at a given instant. The domain of these three places is $Class\_Weight$. It is the same for places *AltitudePossibleVal*, *Altitude* and $Class\_Altitude$ domain and for places *SpeedPossibleVal*, $Speed\_Right\_Wheel$, $Speed\_Left\_Wheel$ and $Class\_Speed$ domain. To legibility reasons the domain of each place is not represented on the figure. The constant values $Altitude\_Limit$ and $Speed\_limit$ correspond to the values that allow to determine if the airplane is on ground, they appear on transition guards.

The application of the tool presented in this paper to this example leads to a partition of the altitude and speed domains into two classes each. The speed and altitude classes are decomposed into two classes each:

$$\begin{aligned} Class\_Altitude &= \{Altitude\_Min..Altitude\_Limit-1\} \\ &\quad \uplus \{\{Altitude\_Limit..Altitude\_Max\} \cup \{Altitude\_Corrupt\}\} \\ Class\_Speed &= \{\{Speed\_Min..Speed\_Limit\} \cup \{Speed\_Corrupt\}\} \\ &\quad \uplus \{Speed\_Limit+1..Speed\_Max\} \end{aligned}$$

To compute these subclasses, we have to fix the constants values *Altitude_Min*, *Altitude_Max*, *Altitude_Limit*, *Altitude_Corrupt*, *Speed_Min*, *Speed_Max*, *Speed_Limit*, *Speed_Corrupt*. The results of the decomposition are independent from these values since we consider that $Speed\_Corrupt = Speed\_Max + 1$ and $Altitude\_Corrupt = Altitude\_Max + 1$. If we don't have this assumption each class is divided into three subclasses $\{Altitude\_Min..Altitude\_Limit-1\} \uplus \{Altitude\_Limit..Altitude\_Max\} \uplus \{Altitude\_Corrupt\}$ and $\{Speed\_Min..Speed\_Limit\} \uplus \{Speed\_Corrupt\} \uplus \{Speed\_Limit+1..Speed\_Max\}$. The designer can modify values of the constants (in the class definition as well as on the guards) without being concerned by the subclasses of its model since they can be automatically computed if necessary.

Though this is not generally the case, the size of the SRG of this system has a constant 783 accessible symbolic states, independently of the size of the domains considered. The number of concrete states represented by this SRG is combinatorial, from 175k RG nodes for domains of cardinality 20 to about $14 \cdot 10^9$ RG nodes for the values extracted from the original specification: 2000 values of speed and 1000 of altitude. The SRG calculation time hardly increases, when the RG tool was saturated for 100 values per domain.

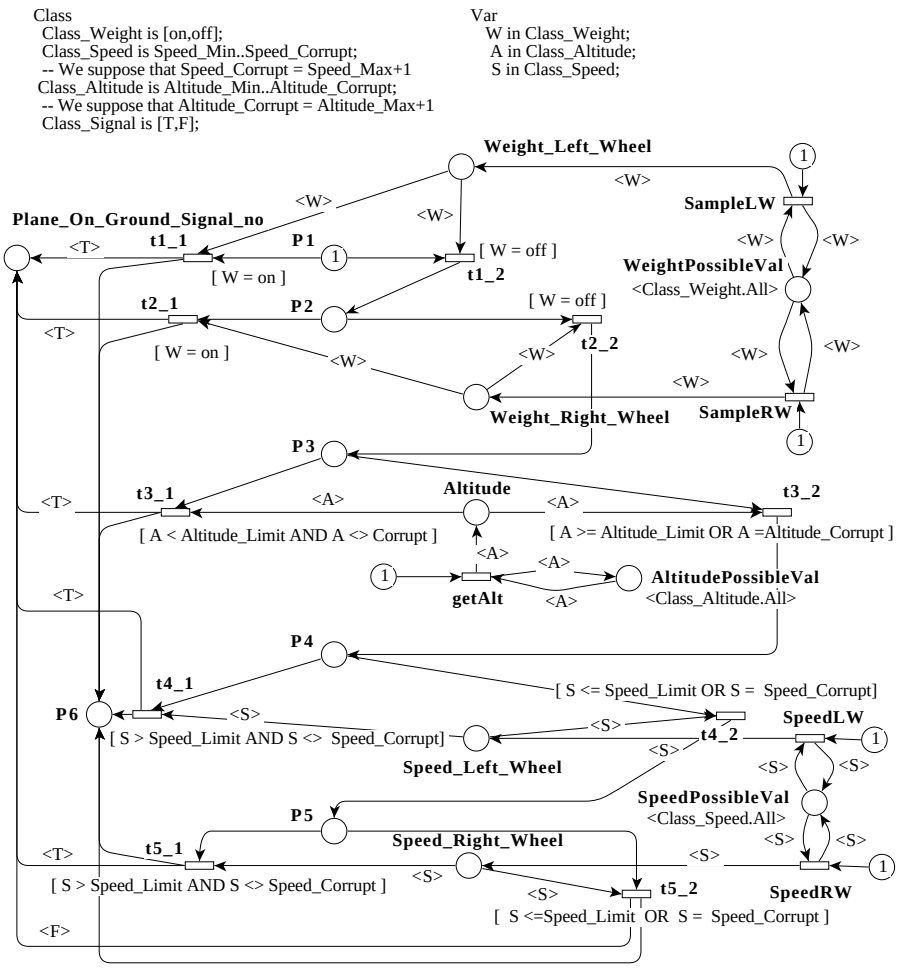

Fig. 1. Petri net model of the function

7 Conclusion

This paper presented a novel algorithm for automatically computing the symmetry break induced by the elements of a class of high-level Petri nets. The resulting information is exploited through translation to Well Formed nets, that may be symbolically analyzed with a symbolic reachability graph. This work has been implemented as a tool for the CPN-AMI software, and relies on GreatSPN for the construction of the SRG. This transformation is transparent for the user, and guarantees that the equivalence classes obtained are minimal with respect to the behavior of the tokens of a net. The analysis is based on the independent computation of the symmetry break induced by each element of a net, and leaves room to apply further structural reductions on the obtained net.

We are currently working at extending this mechanism to optimize the calculation of ESRG, by only taking into account the reachable elements of a net in a given state. This may be done by limiting the computation to the asymmetry induced by a current marking, the possibly fireable transitions for this marking, and their connected arcs. Such an approach will allow symbolic analysis of partially symmetric systems. This work is further used in the development of **L***f***P**, a high-level specification language, that relies on the generation of Petri nets for verification of properties. Another extension in progress is to integrate in the calculation of allowed symmetries the effect of a given LTL or CTL formula, thus allowing property-based model checking. We are further interested in trying to apply a similar exploration technique to automatically deduce symmetries of other formalisms.

References

1. CPN-AMI: a Petri net based CASE environment. url : `http:\/\/www-src.lip6.fr\ /cpn-ami`.
2. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed colored nets and symmetric modeling applications. *IEEE Transactions on Computers*, 42(11):1343–1360, 1993.
3. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. A symbolic reachability graph for coloured Petri nets. *Theoretical Computer Science*, 176(1–2):39–65, 1997.
4. G. Chiola and G. Franceschinis. Structural colour simplification in Well-Formed coloured nets. In *Proc. 4th Int. Workshop on Petri Nets and Performance Models*, pages 144–153, Melbourne, Australia, December 1991.
5. C.N. Ip and D.L. Dill. Better verification through symmetry. In D. Agnew, L. Claesen, and R. Camposano, editors, *Computer Hardware Description Languages and their Applications*, pages 87–100, Ottawa, Canada, 1993. Elsevier Science Publishers B.V., Amsterdam, Netherland.
6. D. Regep, Y. Thierry-Mieg, F. Gilliers, and F. Kordon. Modélisation et vérification de systèmes répartis : une approche intégrée avec **L***f***P**. In *AFADL 2003,Approches Formelles dans l'Assistance au Dveloppement de Logiciels*. INRIA, proceedings, January 2003.
7. M. Doche, I. Vernier-Mounier, and F. Kordon. A modular approach to the specification and validation of an electrical flight control system. In *FME'01, Formal Methods for Increasing Software Productivity*, pages 590–610, Berlin, Germany, March 2001. Springer Verlag.
8. GreatSPN: GRaphical Editor, Analyzer for Timed, and Stochastic Petri Nets. url : `http: //www.di.unito.it/~greatspn/`.
9. J-C. Fernandez, C. Jard, T. Jeron, and C. Viho. An experiment in automatic generation of test suites for protocols with verification technology. *Science of Computer Programming*, 29(1-2):123–146, 1997.
10. Serge Haddad, Jean Michel Ilie, M. Taghelit, and B. Zouari. Symbolic reachability graph and partial symmetries. In *Application and Theory of Petri Nets*, pages 238–257, 1995.
11. D. Poitrenaud and J-F. Pradat-Peyre. Pre- and post-agglomeration for ltl model-checking. *Lecture Notes in Computer Science*, 1825:387–408, 2000.
12. D. Regep and F. Kordon. **L***f***P**: a specification language for rapid prototyping of concurrent systems. In *12th IEEE International Workshop on Rapid System Prototyping*, June 2001.
13. V. Rusu, L. du Bousquet, and T. Jéron. An approach to symbolic test generation. In *2nd International Workshop on Integrated Formal Method (IFM'00)*, number 1945 in LNCS, pages 338–357, Dagstuhl, Germany, 2000. Springer-Verlag.