INT. J. PROD. RES., 15 JULY 2004, VOL. 42, NO. 14, 2741–2756

Taylor & Francis
Taylor & Francis Group

The tool TINA – Construction of abstract state spaces for Petri nets and Time Petri nets

B. BERTHOMIEU*, P.-O. RIBET and F. VERNADAT

In addition to the graphic-editing facilities, the software tool *Tina* proposes the construction of a number of representations for the behaviour of Petri nets or Time Petri nets. Various techniques are used to extract views of the behaviour of nets, preserving certain classes of properties of their state spaces. For Petri nets, these abstractions help prevent combinatorial explosion, relying on so-called partial order techniques such as covering steps and/or persistent sets. For Time Petri nets, which have, in general, infinite state spaces, they provide a finite symbolic representation of their behaviour in terms of state classes.

1. Introduction

Tina (TIme Petri Net Analyser)[1] is a software environment for the editing and analysis of Petri net and Time Petri net (Merlin and Farber 1976). In addition to the usual editing and analysis facilities of such environments (computation of marking reachability sets, coverability trees, semi-flows), *Tina* offers various abstract state space constructions that preserve specific classes of properties of the concrete state spaces of the nets. These classes of properties may be general properties (reachability properties, deadlock freeness, liveness), specific properties relying on the linear structure of the concrete space state (linear time temporal logic properties, test equivalence), or properties relying on its branching structure (branching time temporal logic properties, bisimulation).

The proposed abstractions operate either on timed systems (represented by Time Petri nets) or on untimed systems (represented by Petri nets). For timed systems, such abstractions are mandatory, as these systems typically admit infinite concrete state spaces. Finite abstractions for their behaviour are obtained by the classical technique of state classes and its recent developments. For untimed systems, computation of abstract state spaces helps to prevent combinatorial explosion. In this case, *Tina* uses reduction techniques based on so-called ‘partial order’ methods, such as persistent (stubborn) sets, covering steps, and their combinations.

Tina would typically be used as the front-end of a model-checker, providing it reduced state spaces on which the desired properties can be checked more efficiently than on the original state space (when available). To offer a complete ‘model-checking’ chain, *Tina* may present its results in a variety of formats, understood by popular model checkers or behaviour equivalence checkers, including *MEC* (Arnold *et al.* 1994), a μ-calculus formula checker, *Aldébaran* (Fernandez and Mounier 1991), and *BCG* (Fernandez *et al.* 1996).

Revision received March 2004
LAAS-CNRS, 7 avenue du Colonel Roche, 31077 Toulouse Cedex, France.
*To whom correspondence should be addressed. e-mail: berthomieu@laas.fr
[1]http://www.laas.fr/tina

International Journal of Production Research ISSN 0020–7543 print/ISSN 1366–588X online
http://www.tandf.co.uk/journals
DOI: 10.1080/00207540410001705257

This paper presents an overview of the capabilities of the tool *Tina*. It is organized as follows. Section 2 briefly discusses the proposed 'classical' analysis techniques (reachability and coverability sets, structural techniques). Section 3 is concerned with untimed systems. It presents the various 'partial order' exploration methods implemented and relates them to the classes of properties preserved. Section 4 discusses abstraction techniques for timed systems. It first recalls the classical state classes technique, which preserves linear time properties of the concrete state space, and then a recent refinement preserving branching time properties. Section 5 describes the interfaces of *Tina*, and its software architecture: user interface, editing facilities, input and output formats, and inter-operability with model checkers.

2. Classical methods

The first group of tools provided by *Tina* comprises the 'classical' constructions for Petri nets, like those of the reachability graph and the coverability graph. These standard constructions will not be described here, since details can be found in textbooks such as Diaz (2001). *Tina* implements a specific technique (not described here) for efficiently checking the boundedness property of nets while computing the marking graph of Petri nets. Construction of the reachability graph stops when a place is found unbounded, or when an optional limit on the number of markings, or on the marking of individual places, has been reached. This last feature may be used for on-the-fly verification of properties reducible to a reachability property.

Coverability graphs allow one to compute unbounded places, their computation relying on the well-known Karp and Miller (1969) algorithm. As these graphs are not unique, several heuristics are proposed to construct them, attempting to minimize either the number of marking classes or the computation time.

Still among classical methods, but in the family of structural methods, *Tina* can compute the generator sets for semi-flows on places or transitions of a net (Diaz 2001). This tool will later be complemented by a group of tools dedicated to structural techniques (exploiting invariants deduced from flows and semi-flows).

3. Partial order techniques

3.1. *Background*

The so-called partial order techniques (see Godefroid 1996 for a survey) constitute the framework of the reduction techniques proposed in *Tina*. These techniques are aimed at preventing combinatorial explosion due to the representation of parallelism by interleaving. Figure 1(left) shows the state space of a system made up of three components executing some action in parallel, the interleaving semantics representing its behaviour by a cube. Since the three events are independent, all execution paths reach the same final state. If n components were involved, the state space would be a hypercube with 2^n vertices and $n \times 2^{n-1}$ edges.

A first family of partial order techniques consists of, under certain conditions, exploring only one path among all equivalent possible paths. This strategy was initially developed by Valmari (1990) with the 'stubborn' sets theory, and generalized by Godefroid and Wolper (1991) with the notion of 'persistent' sets. In the case of persistent sets, only a subset of the enabled transitions is examined; the derived graph is then a subgraph of the whole graph.

An alternative approach is that of the 'Covering steps' of Vernadat *et al.* (1996) and Vernadat and Ribet (2003). In that approach, all enabled transitions are

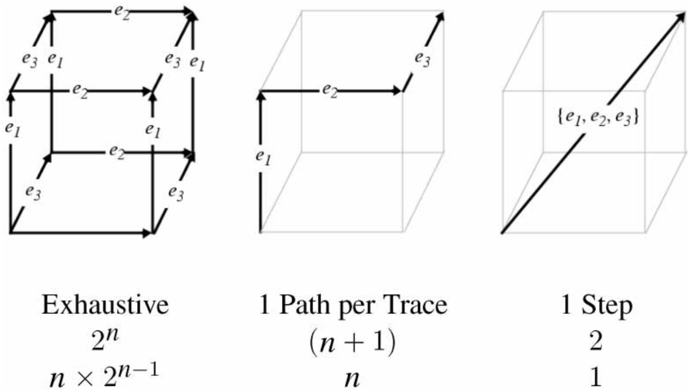


Figure 1. Behaviour graphs for three independent events.

considered, but independent events are grouped into transition steps. Firing transition steps is atomic. The resulting graph is referred to as the Covering Step Graph (CSG).

The benefits obtained in the case of n parallel events are summarized in figure 1. In an exhaustive exploration, both the number of vertices and edges are exponential, as already seen. When exploration is reduced to a single path, the exponential factor disappears. The latter method (covering steps) is optimal in that case, as the number of vertices (and edges) is independent of the number of concurrent events.

'Partial Order' reductions take advantage of an independence relation between events. In practice, the exact relation is not available, as computing it requires building the exhaustive state space first. Therefore, implementations typically rely on approximations of the independence relation. Such approximations may be computed by static analysis of the various formal system descriptions (place/transition net, variable/transition system, synchronized automaton, etc.) (Godefroid and Wolper 1991). *Tina* uses place/transition nets and approximates the independence relation by structural independence, defined as follows: t_1 and t_2 are independent iff $Pre(t_1) \cap Pre(t_2) = \emptyset$ (i.e. transitions t_1 and t_2 share no input place).

A reduction technique is characterized by the compression factor it offers, but also by the analysis power it provides. 'Partial order' techniques are general techniques which can be specialized to preserve specific classes of properties (Godefroid 1996); reduction strategies depend upon the class of properties to be preserved. *Tina* proposes three kinds of reductions: the first kind allows one to verify general properties (absence of deadlock, liveness), the other two are devoted to the verification of specific properties relying on the linear structure or branching structure of state spaces, respectively. For properties concerning the linear structure, two methods ensure preservation of linear time temporal logic properties and linear behavioural equivalence, respectively. For properties relying on the branching structure, a single abstraction is proposed, preserving weak bisimilarity.

The techniques proposed by *Tina* will be illustrated by Milner's scheduler example (Milner 1985). The Petri net figure 2 represents the complete system (scheduler + n sites): n sites cyclically execute action a_i then action b_i. A scheduler constrains the execution of the whole system in such a way that the n sites perform actions $a_1, a_2, \ldots, a_n$ in that order, repetitively. Figure 3 shows the behaviour of the

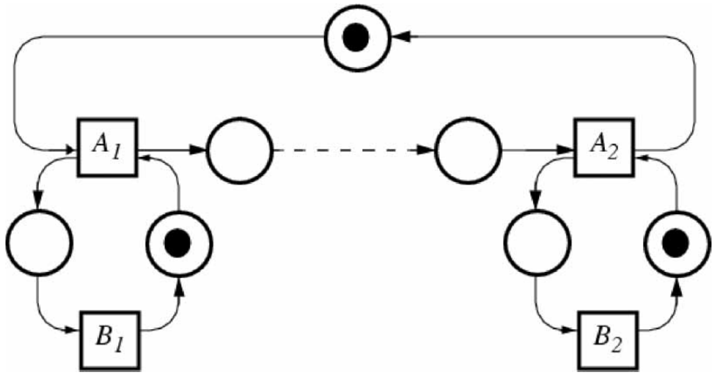


Figure 2. Milner's scheduler for $n = 2$.

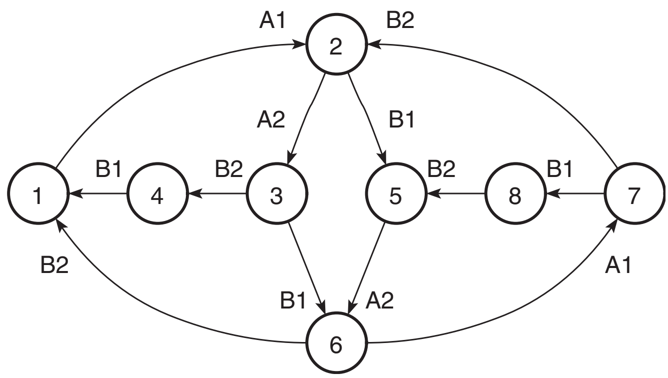


Figure 3. Milner's scheduler – Exhaustive *LTS* for $n = 2$.

system, for two sites, as a labelled transitions system (*LTS*). For n sites the exhaustive *LTS* has $n \times 2^n$ vertices and $(n^2 + n) \times 2^{n-1}$ edges.

3.2. *Preservation of deadlocks*

This section describes the three methods *Tina* offers for computing a reduced *LTS* possessing all the deadlock states present in the exhaustive *LTS*.

3.2.1. *Persistent sets*

A set E of transitions is persistent in a state s iff all transitions not in E that are enabled in s or in states reachable from s by firing transitions not in E are independent of all transitions in E (Godefroid 1996).

A persistent set E contains at least all transitions that have to be explored from a specific state s in order to discover all potential deadlocks: no transition in E can be disabled by firing a sequence of transitions not in E. Note that the set of transitions enabled at s is always persistent in s.

The reduced graph is computed by adding the following rules to any classical enumeration algorithm: *for each reached state s, compute a persistent set associated with s and explore only transitions from this set*.

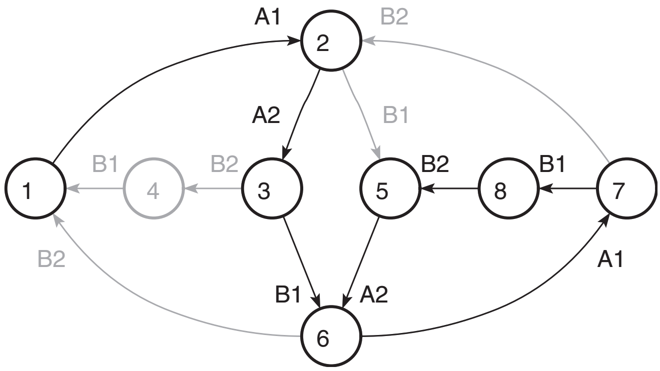


Figure 4. Milner's scheduler – Persistent graph for $n = 2$.

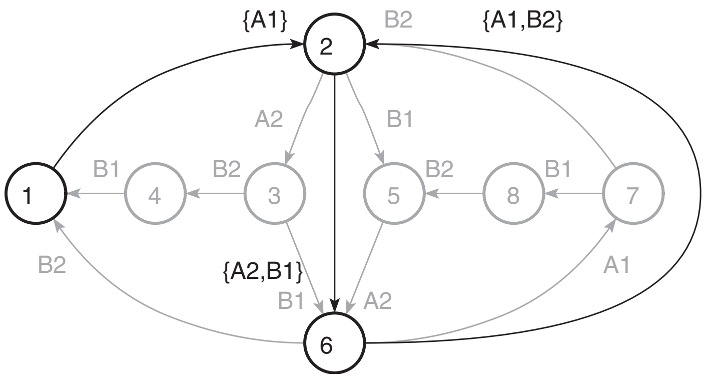


Figure 5. Milner's scheduler – Covering step graph for $n = 2$.

Figure 4 shows a possible reduced state space for the scheduler example with two sites (the exhaustive state space is kept in the background). In the initial state 1, only one transition, A_1, is enabled and so the sole persistent set is $\{A_1\}$. In state 2, two transitions, B_1 and A_2, are enabled and independent: therefore, we can choose either the persistent set $\{B_1\}$ or the set $\{A_2\}$; the second was chosen.

3.2.2. *Covering step graph*

Covering step graphs (*CSG*) were introduced in Vernadat *et al*. (1996). The states of a *CSG* are states of the exhaustive graph, but transitions are steps, i.e. sets of independent transitions. If there exists a step from a state s to a state s' in the *CSG*, then all sequences made of all transitions in the step are firable from s and lead to s'. *CSG* further obey a covering property: every sequence in the exhaustive graph can be extended so that it is covered by a step sequence in the *CSG*. Figure 5 shows a possible covering step graph for the scheduler example. To illustrate the covering property, note that, for example, the sequence $A_1.A_2.B_2.B_1$, extended by A_1, is covered in the *CSG* by the step sequence $\{A_1\}.\{A_2, B_1\}.\{A_1, B_2\}$.

CSG preserve global reachability properties such as liveness and existence of deadlock, and also maximal traces (modulo Mazurkiewicz's trace equivalence (Mazurkiewicz 1986)). Any LTS may be seen as a *CSG* by taking the empty independence relation.

3.2.3. *Persistent step graphs*

The technique of persistent steps (*PSG*) is a specialization of covering steps especially devoted to the sole preservation of deadlock states (Ribet *et al.* 2002). It combines the persistent sets and the transition steps methods. In each state a persistent set is chosen, and then transition steps are computed on this subset of transitions.

Ribet *et al.* (2002) show that persistent step graphs generalize both persistent set and covering step graphs. Like the technique of persistent sets, the persistent steps technique requires a strategy to choose persistent sets in each explored state. Ribet *et al.* (2002) also show that there is no instance of the persistent steps method which is better in all cases than the method of persistent sets. In other words, a particular exploration based on persistent sets only (without steps) can be better than all strategies using steps. In practice, the graph obtained by the *PSG* construction is smaller than the graphs obtained with both the persistent sets and covering steps constructions. Some computing experiments are reported in the next section.

3.2.4. *Computing experiments*

Table 1 shows the number of states obtained with the exhaustive, persistent sets, *CSG*, and *PSG* methods for different examples. All values were computed by *Tina* on a SunBlade 100 Unix workstation, except for the exhaustive case of the Scheduler example, which was computed analytically.

In all reduction cases, the exponential factor disappears. It is interesting to note that, in some examples (e.g. Philosophers), the persistent set construction yields a smaller graph than the *CSG* construction, whereas, in other examples (e.g. Scheduler), the *CSG* construction performs better. The *PSG* results are as good as the best result obtained with persistent sets and covering steps.

Table 2 shows computational results for the swimming pool example of Fribourg and Bérard (1999), also shown in the figure. For this example the *PSG* performs better, in state counts and computation time, than the other two techniques.

3.3. *Preservation of linear structure*

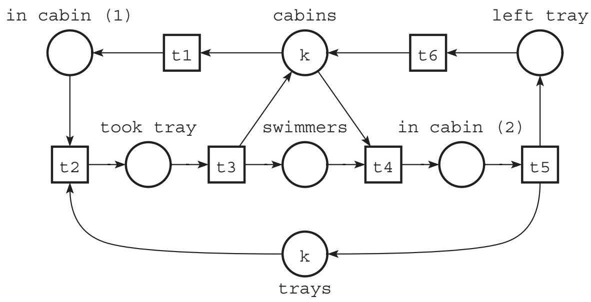
The constructions presented in this section preserve the linear structure of state spaces. Two instances are discussed: the first preserves (linear) behavioural equivalence, and the second preserves linear time temporal logic formulas.

3.3.1. *Failure semantics*

A specialization of the *CSG* was given by Vernadat and Michel (1997) that preserves 'failure semantics' (Van Glabbeck 1990). The construction is parameter-

	Model	Exhaustive	Persistent	*CSG*	*PSG*
1	Scheduler 300	$\approx 6 \times 10^{92}$	1394	301	301
2	Philosopher 8	103 681	233	31 231	227
3	Data base 10	196 831	191	31	31
4	Token Ring 10	35 840	99	52	51
5	Manufacturing system	2034	455	979	360

Table 1. Comparing reduction methods.



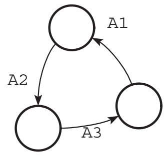
k	Persistent		*CSG*		*PSG*	
10	857	0:00:00	367	0:00:00	87	0:00:00
235	4 602 707	6:40:00	2 19 742	0:01:03	2112	0:00:01
500			9 97 517	0:21:10	4497	0:00:02
600					5397	0:00:02
2×10^5					$\approx 1.8 \times 10^6$	0:16:26

Table 2. Swimming pool example.

ized by a set of observable transitions. Preservation of the linear structure of the state space requires two additional conditions on the basic *CSG* construction:

– a step of the transition must be either reduced to a single observable transition or only composed of unobservable transitions,
– a step cannot contain a transition conflicting with an observable transition.

Let us take the previous scheduler example, and consider observable all synchronization actions between the scheduler and the sites, so $T_{\mathrm{Obs}} = \{A_i : i \in \mathit{Sites}\}$. The resulting *LTS* can be used to show that each site is alternatively scheduled. Moreover, since failure semantics is divergent-sensitive, it is also possible to verify that, always, action A_i eventually happens after action A_{i-1}.



The figure on the left depicts the minimal equivalent *LTS* produced for $n = 3$. In the general case, the minimal equivalent *LTS* consists of n states and n arcs. The size of the *CSG* preserving failure semantics is quadratic, with $n^2 + n$ states and $2 \times n^2$ edges, whereas the size of the exhaustive graph is exponential.

3.3.2. *Linear time temporal logic* $\mathrm{LTL}_{-\mathrm{X}}$

Considering the specific set of atomic variables occurring in a *LTL* formula, we can define a subset of transitions which are 'significant' with respect to the truth value of the formula. According to these significant transitions, a *CSG* specialization for LTL_{-X} (*LTL* without a next-time operator) is proposed which is stuttering equivalent to the exhaustive graph. This means that any stuttering formula defined with these atomic variables is preserved by the reduced graph. Because any formula of LTL_{-X} (Peled 1998) is stuttering-invariant, this LTL_{-X} specialization (Ribet *et al.* 2003) of *CSG* can be employed to verify a formula of LTL_{-X}.

For Milner's scheduler example, consider the property that, from any reachable state, the scheduler will always come back to its initial marking M_1. This property can be written by an LTL_{-X} formula: $\Phi = \Box(\Diamond M_1)$. The exhaustive graph for 10 sites has 10 240 states. Because the marking of place M_1 is only modified by transitions A_1 and A_{10}, a reduced graph can be computed considering these transitions as 'significant'. The property Φ can then be verified on this reduced graph, which only has 15 states.

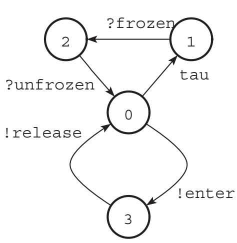
In general, *LTL* formulas are verified by checking that the language of the synchronized product of the automaton representing the system with the Bucchi automaton of the negated formula is empty. If the formula is an LTL_{-X} formula, then, instead of using the full automaton representing the system, one can use any abstraction of it preserving the validity of the formula to be verified. Reducing the automata to be synchronized will result in smaller synchronized automata. For our example, for instance, the synchronized graph obtained from the exhaustive graph has 20 186 states, whereas that obtained from the reduced graph has only 26 states. Further, the truth value of the formula may be checked on the fly while computing the synchronized product.

3.4. *Preservation of branching structure*

Vernadat *et al.* (1996) proposes a specialization of the *CSG* that preserves weak bisimulation. Observational equivalence is defined with respect to a set of observable events. Preservation of the branching structure of the state space requires three additional conditions on the basic CSG construction:

- transition steps are computed only on 'conflict-free' transitions (independent of all others),
- transition steps must contain, at most, one observable transition,
- in order not to lose the branching structure, sub-steps containing only unobservable transitions must be added.

To illustrate this abstraction, consider the data base system presented by Valmari (1989). This system consists of $n \geq 2$ managers and a mechanism ensuring mutual exclusion for critical operations. Each manager may either enter the critical operation and then release the other managers or be frozen by another manager – performing the critical operation – then unfrozen. We consider a local observation where observable events are those of a specific manager (!enter, !release, ?frozen, ?unfrozen).



The minimal weakly bisimilar *LTS* according to this observation is presented on the left. Note the presence of the characteristic 'silent event' τ from the initial state. Since the observation is local, the size of the minimal weakly bisimilar *LTS* is constant (four states and five edges), the size of the equivalent step graph obtained is quadratic in n, while the size of the exhaustive graph is exponential ($\approx n \times 3^{n-1}$ states and $n^2 \times 3^{n-2}$ edges).

4. State class graphs of Time Petri nets

4.1. *Time Petri nets, states, state classes*

Time Petri nets are Petri nets in which a non-negative real interval $I_s(t)$, with rational end-points, is associated with each transition t of the net (Merlin and Farber 1976). Function I_s is called the *Static interval* function.

A state of a TPN is a pair $s = (m, I)$, where m is a marking and I is a function called the *interval* function. Function $I : T \rightarrow I^+$ associates a real non-negative temporal interval with every transition enabled at m. Initially, $s_0 = (m_0, I_0)$, with $I_0(t) = I_s(t)$ for every transition enabled at the initial marking m_0. $Min(I(t))$ and $Max(I(t))$ denote the lower and upper end-points of interval $I(t)$, respectively (when $I(t)$ is not upper-bounded, we let $Max(I(t)) = \infty$).

States evolve as follows: assume the current state is $s = (m, I)$, t is enabled at m, and became enabled for the last time at time τ. Then t cannot fire before time $\tau + Min(I(t))$, and must fire no later than $\tau + Max(I(t))$, except if firing another transition before t made t no longer enabled. Firing transitions takes no time.

This rule defines on the state set a timed reachability relation denoted $\xrightarrow{t@\theta}$. One has $s \xrightarrow{t@\theta} s'$ if transition t may fire from state s at relative time θ; we then say that t is *firable* from s at θ. The state space of a Time Petri net is the set of states reachable from its initial state s_0, equipped with the timed reachability relation $\xrightarrow{t@\theta}$. The relation $(\exists\theta)(s \xrightarrow{t@\theta} s')$ is abbreviated $s \xrightarrow{t} s'$.

A firing schedule is a sequence $(t_i@\theta_i)_{1 \le i \le n}$ of successively firable timed transitions. Its *support* is the sequence of transitions $t_1 \ldots t_n$. The *firing domain* of a state (m, I) is the set of vectors $\{\underline{\phi} | (\forall k)(\underline{\phi}_k \in I(k))\}$, with their components indexed by the transitions enabled at m.

As transitions may fire at any time in their temporal intervals, the states of a Time Petri net generally admit an infinity of successors by the timed reachability relation. Any finite representation of this state space must thus rely on some agglomeration of states. These agglomerations are called *state classes*.

In their most general definition, state class spaces are covers of the state space equipped with a transition relation satisfying property (EE) below (c and c' denote state sets). Further, it is assumed that all states in a state class bear the same marking.

$$\text{(EE)} \quad (\forall t \in T)(\forall c, c')(c \xrightarrow{t} c' \Leftrightarrow (\exists s \in c)(\exists s' \in c')(s \xrightarrow{t} s'))$$

In this framework, several different state class spaces may be defined from a single state space, depending on the properties of the state spaces the agglomeration preserves. Note that transitions between state classes are no longer timed, and state classes and their reachability relation allow one to abstract time from the behaviour of a net. The tool *Tina* proposes several state class space constructions, preserving either the properties of the state space one can express in linear time temporal logics (such as *LTL*), or those expressed in branching time temporal logics (such as *CTL*).

To enable a synthetic definition of the state class spaces investigated, it is convenient to first introduce characteristic systems: for every firing sequence σ, the (relative) times at which transitions in the sequence may fire (variables $\underline{\theta}$), and the state reached (described by its firing domain, variables $\underline{\phi}$), are related by an inequality system of the following form, called the *characteristic system* of sequence σ:

(1) $P\underline{\theta} \le \underline{p}$,
(2) $\underline{0} \le \underline{\phi}, \underline{e} \le \underline{\phi} + M\underline{\theta} \le \underline{l}$, with $\underline{e}_k = Min(I_s(k)), \quad \underline{l}_k = Max(I_s(k))$.

Subsystem (1) describes the vectors of possible relative firing times $\underline{\theta}$ for all transitions in σ. For every such $\underline{\theta}$, subsystem (2) describes the firing domain of the state reached.

These systems, readily computed (Berthomieu and Vernadat 2003), can be presented as a tree KG rooted at K_ϵ. System K_σ characterizes the set of states (generally infinite) reachable from the initial state by firing schedules of support σ. Conversely, to the state graph of the net, tree *KG* is finitely branching, but it may still be infinite, as it has as many nodes as there are firable sequences.

4.2. *Preserving* LTL *properties, linear state classes*

4.2.1. *Linear state classes, construction* LSCG

Concerning Time Petri nets, the first state class graph construction provided by *Tina* is the classic one introduced by Berthomieu and Menasche (1983), and further accounted for by Berthomieu and Diaz (1991). It can be explained as follows.

For every firing sequence σ, let C_σ be the set of states reachable by schedules of support σ (as characterized by characteristic system K_σ). Markings and firing domains are extended to such sets of states as follows: the marking of C_σ is the marking of any of its component states (recall that all states in a class bear the same marking), and the firing domain of C_σ is the union of the firing domains of all states constituting C_σ. Consider now the equivalence relation $\cong$ satisfied by two such state sets when they have the same marking and firing domain.

The *linear state class graph* (*LSCG*) is the set of sets C_σ, for all firable sequences σ, considered modulo equivalence $\cong$, and equipped with the transition relation: $c \xrightarrow{t} c'$ iff $(\exists s \in c)(\exists s' \in c')(s \xrightarrow{t} s')$.

The *LSCG* coincides with the graph obtained from the tree of characteristic systems by identifying nodes equivalent by $\cong$ (that is, systems with equal solution sets after elimination of the $\underline{\theta}$ path variables). A direct construction was proposed by Berthomieu and Menasche (1983). The *LSCG* of the net presented in figure 6 admits 83 classes and 160 transitions.

The *LSCG* preserves those properties of the state graph of the net expressible as formulas of linear time temporal logics (such as *LTL*), hence its name. It can be shown that, when two characteristic systems K_σ and $K_{\sigma'}$ are equivalent by $\cong$, with σ and σ' leading to the same marking, then the subtrees of *KG* they define are isomorphic. This implies preservation by *LSCG* of all traces and maximal traces of the state graph, and thus of *LTL* properties.

4.2.2. *Linear state classes with multi-enabledness, construction* $\mathrm{LSCG_m}$

In the *LSCG* construction, every enabled transition is associated with exactly one temporal variable describing the firing domain. In addition to the basic *LSCG* construction, *Tina* provides a variant introduced by Berthomieu (2001) in which self-concurrent, or multiply enabled, transitions are associated with as many temporal variables as there are enabled instances of the transition.

This interpretation has several practical uses, notably when the presence of tokens in the places of the net is interpreted as the arrival of events. The treatment

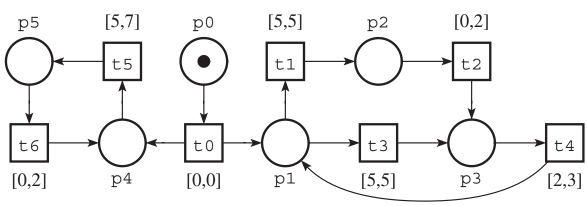


Figure 6. A Time Petri net.

of Berthomieu (2001), consisting of ordering enabling instances of self-concurrent transitions according to their date of birth, may also help to handle symmetry of the state space: this makes it possible to model k similar processes by a single net, the marking of which is parameterized by k. This contributes to the reduction of the state space. The construction preserves the linear time temporal properties of the state graph.

4.2.3. *Strong linear state class graphs, construction* SSCG

A third construction also preserving *LTL* properties is proposed. As in section 4.2.1, let C_σ be the set of states reachable by firing schedules of support σ. The *strong linear state classes* are exactly those sets C_σ that are not considered modulo equivalence $\cong$, but simply modulo their natural set equality. These classes yield the *strong state class graph* (*SSCG*).

A construction of the *SSCG* is given by Berthomieu and Vernadat (2003). Strong linear classes are represented by a marking associated with an inequality system expressed in terms of 'clock domains' rather than firing domains. The clock $\underline{\gamma}_t$ associated with the enabled transition t is the time elapsed since t was last enabled. The clock system associated with strong class C_σ coincides with subsystem (1) of the characteristic system K_σ, with equations $\underline{\gamma} = M\underline{\theta}$ added ($\underline{\gamma}$ are the clock variables, bijectively associated with the enabled transitions), and then variables $\underline{\theta}$ eliminated.

Clock vectors denote states. It is shown by Berthomieu and Vernadat (2003) that an equivalence relation $\equiv$ can be computed for strong classes represented by a marking and a clock domain such that $C_\sigma \equiv C_{\sigma'}$ iff they denote equal sets of states. Briefly, if all enabled transitions have bounded temporal intervals, then equivalence $\equiv$ coincides with equality of the solution sets of the clock systems. When this is not the case, an extra operation, called relaxation, has to be performed on clock systems, before their comparison.

Like the *LSCG*, the *SSCG* preserves *LTL* properties, but it is typically larger than the *LSCG* in terms of number of classes, and its computation is more expensive. The *SSCG* construction would thus be a poor replacement for the *LSCG*. In fact, it is only provided because it constitutes the starting point of the construction preserving branching time properties, described in the next section. The *SSCG* of the net presented in figure 6 admits 107 classes and 205 transitions.

4.3. *Preservation of* CTL* *properties, atomic state classes (*ASCG*)*

Branching properties are those expressed in branching time temporal logics such as *CTL* or *CTL**, or in modal logics like *HML* or the μ-calculus. In the absence of silent transitions, it is known that these properties are preserved by the bisimulation. Any graph of classes bisimilar with the state graph of the net (time information omitted for the latter) preserves all its branching properties. This section addresses constructions that yield a state class graph bisimilar with the state graph.

A first such construction was proposed by Yoneda and Ryuba (1998). *Tina* offers an alternative construction introduced in Berthomieu and Vernadat (2003), under the name *atomic state class graph* (*ASCG*). As for computing bisimulations on an *LTS*, this graph is built by a technique similar to the 'partition refinement' technique of Paige and Tarjan (1987) and Tripakis and Yovine (1996). A class is said to be atomic, or stable, if it is atomic versus any other class. A class is atomic versus another if each of its states has a successor state in the latter or none has. The graph of atomic state classes is obtained by refinement of the strong state class graph discussed in section 4.2.3: its classes are partitioned until all of them are atomic.

Technically, every unstable class is partitioned by computing linear constraints, non-redundant in its clock system, and such that the constraints are necessary for a state to have a successor in the target class considered.

This construction is generally expensive (though finite, the number of necessary class splits may be very large), but it allows verification of the largest set of properties. Some computational results will be presented in section 4.5. Atomic state classes are represented like strong state classes, that is as a marking associated with an inequality system on the clock space. The *ASCG* of the net presented in figure 6 admits 101 classes and 431 transitions.

4.4. *Preservation of quantitative temporal properties*

Finally, there is a class of properties of great practical interest, but for which no dedicated construction is proposed by *Tina*: that of 'quantitative' temporal properties as may be expressed in, for example, the *TCTL* logic.

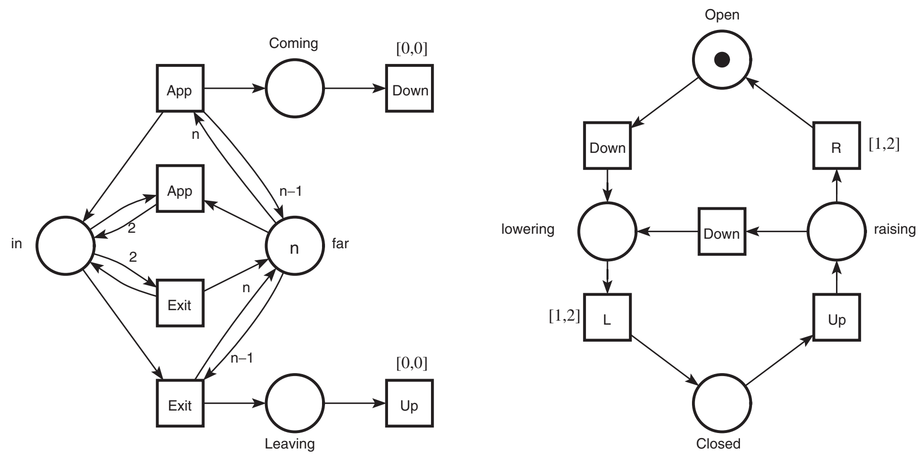
Although no dedicated support is provided for proving such properties, many of them can be verified using the standard technique of observers. This technique consists of encoding a 'quantitative' property p of the net into a qualitative (*LTL* or *CTL**) property p' of the net augmented with an auxiliary 'observer' component, in such a way that p' holds in the augmented net if and only if p holds in the original net. Then, to check p, it suffices to invoke the construction of *Tina* that preserves the intended group of properties on the augmented net, and prove p' on that abstraction. The technique is applicable to a large class of properties, notably those reducible to reachability properties.

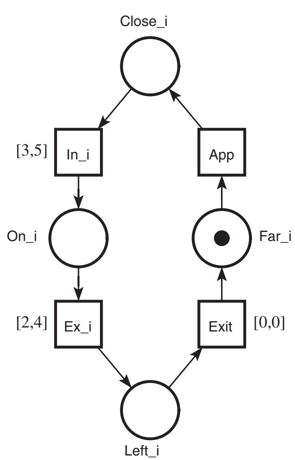
4.5. *Example and comparisons*

Figure 7 shows a Time Petri net version of the classical level crossing example (Bérard *et al.* 2001). The net models are obtained by the parallel composition of n train models (lower left), synchronized with a controller model (upper left, n instantiated), and a barrier model (upper right). The specifications mix timed and untimed transitions (those labeled *App*).

For each model, we constructed with *Tina* its *LSCG*, *SSCG* and *ASCG*. Sizes and computing times obtained on a SunBlade 100 Unix workstation are also shown in figure 7. Computing times for the *ASCG* are much longer than those for the *LSCG* or *SSCG*, but that construction preserves more properties. Safety properties such as 'the barrier is closed when a train crosses the road' can be checked on any graph, but liveness properties such as 'when no train approaches, the barrier eventually opens' must be checked on the *ASCG*. Temporal properties such as 'when a train approaches, the barrier closes within some given delay' generally translate to safety or liveness properties of the net composed of 'observer nets' deduced from the properties.

Note the fast increase in the number of classes with the number of trains, for all constructions. For this particular example, this number could be greatly reduced by exploiting the symmetries of the state space resulting from replication of the train model. For linear state classes, an alternative is allowed by the variant $LSCG_m$ of the *LSCG* construction discussed in section 4.2.2, in which a transition enabled k times is associated with k distinct intervals, instead of one. For our example, symmetries





n		$LSCG$	$SSCG$	$ASCG$
	$Classes$	11	11	12
1	$Edges$	14	14	16
	$CPU(s)$	0.00	0.00	0.00
	$Classes$	123	141	196
2	$Edges$	218	254	859
	$CPU(s)$	0.01	0.04	0.20
	$Classes$	3101	5051	6981
3	$Edges$	7754	13019	49997
	$CPU(s)$	0.70	2.53	52.54
	$Classes$	134501	351271	?
4	$Edges$	436896	1193376	?
	$CPU(s)$	85.50	972.46	?

Figure 7. Level crossing example.

could be handled by modelling trains by a single copy of the train model shown in figure 7, marked by *n*, instead of *n* copies, and using that interpretation of multi-enabledness.

5. User interface

The *Tina* toolbox is implemented in a modular way. These modules can be used independently or in combination. Modules include:

– a graphic editor for Petri nets, Time Petri nets, or automata, including automatic drawing facilities;
– a tool for building state space abstractions, implementing all the constructions presented in the previous sections; and
– a structural analysis tool (in progress).

Used alone, the editor produces files that later can be read by the state space construction and structural analysis tools. But these tools may also be invoked without leaving the editor, and the editor is able to edit and draw their outputs.

Used alone, the construction and analysis tools behave like filters. This eases their insertion into specific or existing development chains. Their command line allows one to select the desired abstraction. They admit as input descriptions in either graphical format (produced by the graphic editor) or textual format (produced by hand or by program), and may produce their results in a variety of formats.

The textual input format is simple and intuitive. Several output formats are available, including a 'verbose' format for pedagogical uses, the automata and *BCG* formats of the tools *Aldébaran* (Fernandez and Mounier 1991) and *BCG* (Fernandez *et al.* 1996) for equivalence analysis, and the *MEC* (Arnold *et al.* 1994) format for checking μ-calculus formulas. In this way, *Tina* may be used as a front end by a number of verification tools, possibly at the expense of writing a simple conversion filter translating one of the available output formats into one accepted by the tool used.

A screen snapshot of a typical *Tina* session is shown in figure 8, with a Time Petri net being edited, the textual result of a behaviour construction, and a graphical representation of the behaviour built.

6. Conclusion

This paper describes *Tina*, a software tool for the editing and analysis of Petri nets and Time Petri nets. In addition to the standard functionalities of such tools (editing, classical reachability and structural analyses), *Tina* proposes the computation of abstract state spaces.

Different abstractions are proposed that preserve various classes of properties: general reachability properties, or specific properties – preserving either the linear or branching structure of the concrete state space – expressed using either temporal logics or behavioural equivalence. Two abstractions based on 'partial orders' and 'state classes' apply to untimed systems and timed systems, respectively. For timed systems, building abstract spaces is mandatory since the concrete state spaces are generally infinite; abstract spaces are finite symbolic representations for the infinite concrete state spaces. For untimed systems, abstract state spaces help prevent combinatorial explosion. In highly concurrent systems, they often result in a drastic size reduction of the state space.

The application domain of *Tina* is wide. *Tina* is being used in several industrial projects; it belongs, for example, to the set of verification tools retained for the RNTL COTRE[2] project. Besides industrial applications, the different constructions proposed by *Tina* make it a very useful tool for education or training.

For the 'partial order' approaches, work in progress concerns the complementarities between the available techniques, notably those between covering steps and 'ample sets' for LTL_{-X} model-checking. For timed systems, techniques to ease verification of quantitative temporal properties (like those expressed in, for example, *TCTL*) are being investigated. Finally, investigations are ongoing concerning the

[2]Composants Temps Réel, Real time components, http://www.laas.fr/COTRE

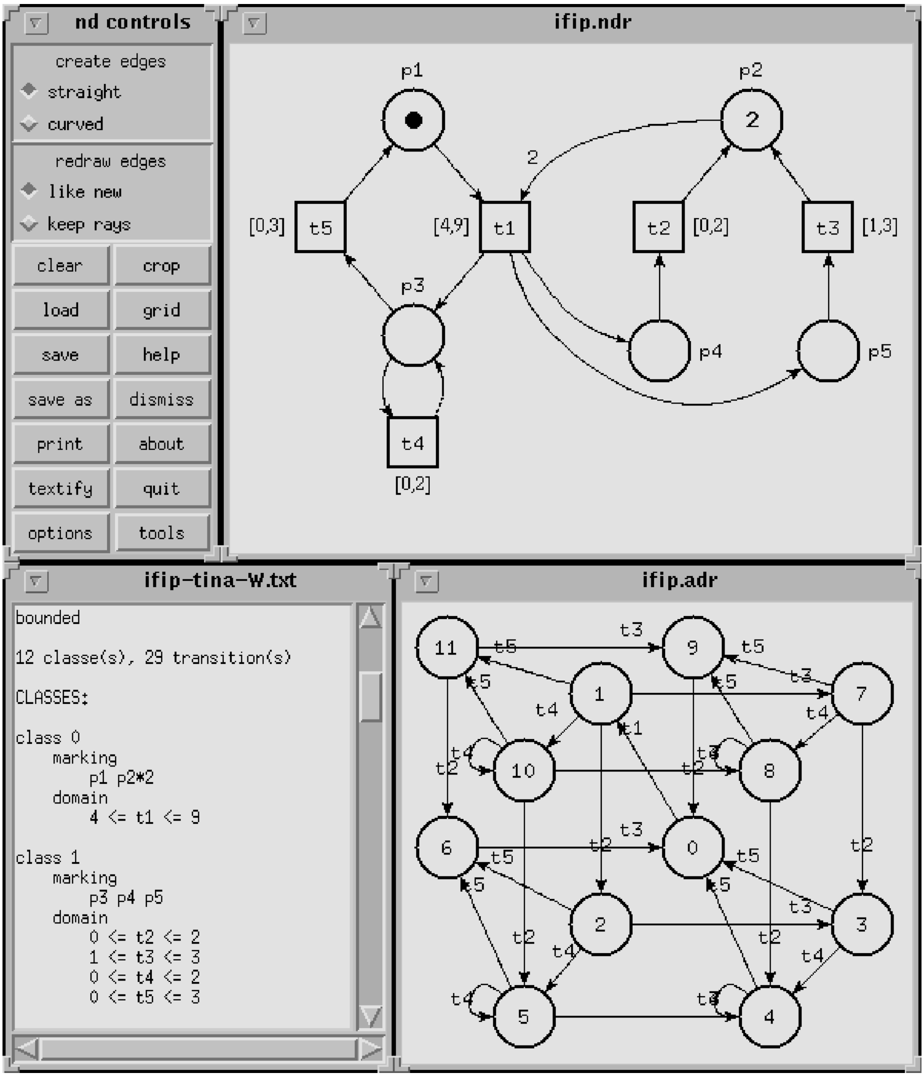


Figure 8. Screen snapshot of a typical *Tina* session.

possible combinations of 'partial order' and 'state classes' techniques in order to compact further the symbolic state spaces of timed systems.

References

ARNOLD, A., BEGAY, D. and CRUBILLÉ, P., 1994, *Construction and Analysis of Transition Systems with MEC* (Singapore: Word Scientific).

BÉRARD, B., BIDOIT, M., FINKEL, A., LAROUSSINIE, F., PETIT, A., PETRUCCI, L. and SCHNOEBELEN, P., 2001, *Systems and Software Verification Model-Checking Techniques and Tools* (Berlin: Springer).

BERTHOMIEU, B., 2001, La méthode des classes d'états pour l'analyse des réseaux temporels – mise en œuvre, extension à la multi-sensibilisation. Proceedings of Modélisation des Systèmes Réactifs, Toulouse, France.

BERTHOMIEU, B. and DIAZ, M., 1991, Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, **17**(3), 259–273.

Berthomieu, B. and Menasche, M., 1983, An enumerative approach for analyzing time Petri nets. Proceedings of IFIP Congress Series, **9**, 41–46.

Berthomieu, B. and Vernadat, F., 2003, State class constructions for branching analysis of time Petri nets. Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, Warsaw, Poland (Berlin: Springer).

Diaz, M., 2001, Méthodes d'analyse des réseaux de Petri. In *Les Réseaux de Petri. Modéles fondamentaux*. Traité IC2 Information-Commande-Communication (Hermes Science).

Fernandez, J.-C., Garavel, H., Kerbrat, R., Mateescu, R., Mounier, L. and Sighireanu, M., 1996, A protocol validation and verification toolbox. Proceedings of 8th Conference on Computer-Aided Verification (Berlin: Springer).

Fernandez, J.-C. and Mounier, L., 1991, A toolset for deciding behavioral equivalences. Proceedings of CONCUR'91 (Berlin: Springer).

Fribourg, L. and Bérard, B., 1999, Reachability analysis of (timed) Petri nets using real arithmetic. Proceedings of CONCUR'99 (Berlin: Springer).

Godefroid, P., 1996, *Partial-Order Methods for the Verification of Concurrent Systems* (Berlin: Springer).

Godefroid, P. and Wolper, P., 1991, Using partial orders for the efficient verification of deadlock freedom and safety properties. Proceedings of Computer-Aided Verification CAV'91 (Berlin: Springer).

Karp, R. and Miller, R., 1969, Parallel program schemata. *Journal of Computer and System Sciences*, **3**(2), 147–195.

Mazurkiewicz, A., 1986, Trace theory, Petri Nets: applications and relationships to other model of concurrency, advances in Petri nets 1986, Part II. Proceedings of an advanced Course (Berlin: Springer).

Merlin, P. M. and Farber, D. J., 1976, Recoverability of communication protocols: Implications of a theoretical study. *IEEE Trans. Comm.* **24**(9), 1036–1043.

Milner, R., 1985, *Communication and Concurrency* (Engelwood Cliffs, NJ: Prentice Hall).

Paige, P. and Tarjan, R., 1987, Three partition refinement algorithms. *SIAM Journal on Computing*, **16**(6), 973–989.

Peled, D., 1998, Ten years of partial order reduction. Proceedings of Computer-Aided Verification CAV'98 (Berlin: Springer).

Ribet, P.-O., Vernadat, F. and Berthomieu, B., 2002, On combining the persistent sets method with the covering steps graph method. Proceedings of FORTE 2002 (Berlin: Springer).

Ribet, P.-O., Vernadat, F. and Berthomieu, B., 2003, Graphe de pas couvrant préservant *LTL_{-X}. Report 03053. LAAS-CNRS.*

Tripakis, S. and Yovine, S., 1996, Analysis of timed systems based on time-abstracting bisimulations. Proceedings of Computer-Aided Verification CAV'96 (Berlin: Springer).

Valmari, A., 1989, Stubborn sets for reduced state space generation. Proceedings of Application and Theory of Petri Nets (Berlin: Springer).

Valmari, A., 1990, A stubborn attack on state explosion. Proceedings of Computer-Aided Verification CAV'90. ACM, DIMACS vol. 3, pp. 25–42.

Van Glabbeek, R. J., 1990, The linear time-branching time spectrum. Proceedings of CONCUR'90 (Berlin: Springer).

Vernadat, F., Azéma, P. and Michel, F., 1996, Covering step graph. Proceedings of ATPN'96 (Berlin: Springer).

Vernadat, F. and Michel, F., 1997, Covering step graph preserving failure semantics. Proceedings of ATPN'97 (Berlin: Springer).

Vernadat, F. and Ribet, P., 2003, Graphes de Pas couvrants: une approche ordre partiel. In *Vérification et mise en oeuvre des réseaux de Petri* (Hermes Science).

Yoneda, T. and RYUBA, H., 1998, CTL model checking of Time Petri nets using geometric regions. *IEEE Transactions on Information and Systems*, **E99-D**(3), 1–10.