ELSEVIER Theoretical Computer Science 202 (1998) 127–161

Theoretical Computer Science

A theory of implementation and refinement in timed Petri nets

Miguel Felder[a], Angelo Gargantini[b], Angelo Morzenti[b,*]

[a]*Facultad de Ciencias Exactas y Naturales, Dpto. de Computacion, Universidad de Buenos Aires, Argentina*
[b]*Politecnico di Milano, Dipartimento di Elettronica e Informazione, Milano, Italy*

Received September 1995; revised May 1996
Communicated by G. Rozenberg

Abstract

We define formally the notion of implementation for time critical systems in terms of provability of properties described abstractly at the specification level. We characterize this notion in terms of formulas of the temporal logic TRIO and operational models of timed Petri nets, and provide a method to prove that two given nets are in the implementation relation. Refinement steps are often used as a means to derive in a systematic way the system design starting from its abstract specification. We present a method to formally prove the correctness of refinement rules for timed Petri nets and apply it to a few simple cases. We show how the possibility to retain properties of the specification in its implementation can simplify the verification of the designed systems by performing incremental analysis at various levels of the specification/implementation hierarchy.

Keywords: Timed Petri nets; Refinement; Temporal logic; Real-time and reactive systems; Requirements; Specification; Design; Implementation; Correctness; Verification

1. Introduction

Real-time systems are required to manage their resources in a way that predictably satisfies some given timing constraints. Such systems are often embedded in critical applications such as patient monitoring systems, plant supervision systems, traffic control systems: their correctness is of primary importance, since their failure can have enormous costs and lead to unrecoverable damages. In the past years, the research on formal methods for the specification and verification of real-time systems has been

*Corresponding author. E-mail: morzenti@elet.polimi.it. The present research was partially supported by projects UBACYT EX 186 and CEE KIT 125.

0304-3975/98/$19.00
PII S0304-3975(97)00078-9

particularly active, especially in the field of temporal logic, resulting in the proposal of several specification formalisms and verification methods.

The proposed models are however rarely employed in the industrial development of such systems, where informal and semiformal methods are still largely prevalent. One of the reasons for this unsatisfactory state of the art is that the systematic or algorithmic analysis techniques are very complex so that they cannot be scaled up to realistic systems. For instance, the algorithms proposed for system verification and validation are often exponential in the size of the specification [3, 10].

Often, however, the final specification of a (real time) system or its high-level design are derived through a sequence of refinement steps. In each of these steps, starting from an abstract system description to be considered as its specification, one derives an "implementation", i.e., a more detailed version that includes elements deriving from design choices but retains the required properties. If these repeated refinement steps are conducted in a systematic, careful way, the verification activity need not be repeated from scratch for each implementation step, since the system can be analyzed incrementally. The overall cost of the verification can thus be kept at a reasonable level by reusing in each step the results already obtained in the preceding phases.

In this paper we address the problem of reducing the overall specification and design effort for real-time system developed through a sequence of refinement steps. We report here the application of these ideas to the case when the real time system is abstractly modeled with timed Petri nets (TPNs for short, a kind of Petri net where each transition is associated with a firing time interval describing its earliest and latest firing time after enabling[1]) and its timing requirements are described by means of formulas written in TRIO (a temporal logic providing a metric on time distances, particularly suitable for specifying real-time systems). In this framework, we formally define the notion of implementation among two timed Petri nets: a net I acting as an implementation implements a net S acting as its specification, if it satisfies all the timing properties that are guaranteed by S. In previous works [11, 12], we defined an axiomatic system for TRIO and an axiomatization of timed Petri nets that adequately copes with the salient features of this operational formalism, such as nondeterministic behavior, multiple simultaneous transition firings, zero-time and infinite-time transitions, and unbounded accumulation of tokens in places. Based on such axiomatization, in the present work we formally characterize properties of TPNs as TRIO theorems describing timing relations among their transition firings, and the notion of implementation among TPNs S and I in terms of a TRIO metatheorem asserting that the theorems holding in the specification net can be proved (under a suitable translation) also in the implementation net. Furthermore we provide methods, based on

[1] In the open literature the term "*time* Petri net" (instead of time *d*) is also used, without any apparent distinction in meaning. Here we use the term timed Petri net because we consider it more appropriate from the standpoint of English usage.

sufficient conditions, to prove in several significant cases that two given nets are in the implementation relation.

The proof of implementation among timed Petri nets, however, may not be performed algorithmically and it may require a certain amount of skill and ingenuity, because no general guideline can be provided for it and the TRIO axiomatization is not complete (we recall that the language includes arithmetic over the temporal domain). In a companion paper [9], we introduced a set of refinement rules for timed Petri nets which allow a designer to substitute a place or transition with a net fragment in such a way that the resulting net is a implementation of the original one. Here we provide systematic methods to formally prove the correctness of such refinement rules. Once the correctness of a rule has been formally proven, it can be applied to particular nets systematically or even automatically [10], since only the topological relations among net elements and the algebraic relations among the time bounds of the involved transitions must be checked.

By adopting this design method based on sequences of refinement steps, one can obtain nets that satisfy by construction all properties specified by the initial abstract version of the system. Moreover, the properties "inherited by refinement" can further be used as lemmas in the more detailed analysis of the final version of the system. In this way, when analyzing an implementation obtained through refinement, the analysis effort can be greatly reduced by performing the proof of intermediate lemmas on the first, more abstract and simple versions of the system.

The notion of refinement of Petri nets has already been studied in the literature [27, 26, 21] but, to the best of our knowledge, with reference only to untimed Petri nets. Here, and in [9], we propose new techniques specifically devoted to real-time systems, thus employing timed Petri nets and a temporal logic with a metric on time such as TRIO. We characterize properties that we wish to be preserved in implementations in a syntactic way, i.e., by means of TRIO formulas, whereas other approaches [9, 28, 29, 14] adopt more semantic characterizations based on execution traces and behaviors. Under this respect, [6] adopts a treatment closer to ours. because it uses the temporal logic MCTL (a modular extension of CTL) to describe properties of Petri net modules; this emphasis on modularity is also a major feature of the above-mentioned work on Petri nets [29].

Other contributions that appeared recently in the literature are less closely related to the present work, in that they study the refinement operation in a different or broader context than (timed) Petri nets, referring to generic state-transition systems or to reactive systems without explicit and quantitative real-time constraints. Our definitions of implementation and refinement, based on the ability to ensure at lower levels in the specification/implementation hierarchy the properties that are specified at the highest specification level, follows similar notions introduced in [2]. The approach proposed in [1] deals generically with any state-based machine, but does not take into account real-time aspects, since it considers just untimed sequences of machine states and focuses on safety and liveness properties; [1] also differs from our work in that the notion of correct refinement is defined in terms of mappings among states or

behaviors, while we refer to properties explicitly expressed through logic formulas. Other approaches to the refinement operation [18, 14] greatly emphasize compositionality and modularity, both in the system structure and in the proof of its properties. The ideas on incremental analysis of refined systems presented in this paper are strongly related to the notions of compositionality and incrementality reported in [30].

The present work is structured as follows. Sections 2 and 3 provide a brief summary of the TRIO language and its use in the axiomatization of timed Petri nets. Section 4 formally defines the implementation relation among timed Petri nets. Section 5 introduces a set of refinement rules and provides methods for proving their correctness. Section 6 presents an example of incremental analysis, and Section 7 draws conclusions.

2. The temporal logic TRIO

TRIO is a temporal logic equipped with operators that provide a metric on time, since they express quantitatively the distance in time between events and the length of time intervals. The underlying time is assumed to be linear and the logic easily accommodates both discrete and dense models of time. In the present paper, we assume as model of time the set of real numbers, which makes the time domain continuous and unlimited both in the past and in the future. The logic is first order and typed, in that every variable has an associated domain of possible values, and every predicate and function has associated domain and range. The language includes time-independent predicates, whose interpretation is independent of the current time instant, and time dependent ones, representing relations that may change with time.[2] The fundamental modal operator *Dist* is defined in such a way that if A is a formula and t is a term of the temporal type, then $Dist(A, d)$ is a formula meaning that A holds at an instant d time units (t.u.) in the future (if $d > 0$) or in the past (if $d < 0$) or at the current time (if $d = 0$).

Several derived temporal operators may be defined starting from Dist, using the propositional connectives, first-order quantification, and conditions on the temporal argument of Dist. A sample thereof is given in Table 1, together with short intuitive explanations, whenever needed.

TRIO has been given a model-theoretical semantics in [16] in a fairly standard way. In [12] we defined a sound and (relatively) complete axiomatic system which is reported in [8], together with some useful metatheorems. In this axiomatic system the metatheorems usually found in ordinary predicate calculus can be proved: we mention, among others, the Deduction Theorem, the Generalization theorem, and the

[2] Elsewhere we defined more complex versions of the language that include time-dependent variables and functions: we ignore here such features which are not essential for the presented results.

Table 1
A sample of derived temporal operators

$Futr(\varphi, t)$	$\stackrel{\text{def}}{=}$	$t \geqslant 0 \wedge \text{Dist}(\varphi, t)$	Future
$Past(\varphi, t)$	$\stackrel{\text{def}}{=}$	$t \geqslant 0 \wedge \text{Dist}(\varphi, -t)$	Past
$Lasts(\varphi, t)$	$\stackrel{\text{def}}{=}$	$\forall t'(0 < t' < t \rightarrow \text{Dist}(\varphi, t))$	φ lasts for a time interval of length t
$SomF(\varphi)$	$\stackrel{\text{def}}{=}$	$\exists t(t > 0 \wedge \text{Dist}(\varphi, t))$	φ occurs sometimes in the future
$AlwF(\varphi)$	$\stackrel{\text{def}}{=}$	$\forall t(t > 0 \rightarrow \text{Dist}(\varphi, t))$	φ holds always in the future
$AlwP(\varphi)$	$\stackrel{\text{def}}{=}$	$\forall t(t - 0 \rightarrow \text{Dist}(\varphi, -t))$	φ always held in the past
$Alw(\varphi)$	$\stackrel{\text{def}}{=}$	$\forall t\ \text{Dist}(\varphi, t)$	φ always holds
$Som(\varphi)$	$\stackrel{\text{def}}{=}$	$\exists t\ \text{Dist}(\varphi, t)$	φ occurs sometimes
$WithinF(\varphi, t)$	$\stackrel{\text{def}}{=}$	$\exists t'(0 \leqslant t' \leqslant t \wedge \text{Dist}(\varphi, t'))$	φ will occur within t time units
$WithinP(\varphi, t)$	$\stackrel{\text{def}}{=}$	$\exists t'(0 \leqslant t' \leqslant t \wedge \text{Dist}(\varphi, -t'))$	φ occurred within the last t time units

Existential instantiation theorem [7]. We also recall here an important and intuitive metatheorem, frequently used in TRIO derivations, that we will apply in Section 6. This is called the Temporal Generalization theorem; it asserts that if $\Gamma \vdash \alpha$ and every formula of Γ is of the type Alw(γ), then $\Gamma \vdash$ Alw(α), i.e., if the hypotheses under which a property is proved are not restricted to the present but hold at any time, then the derived property is also always true.

3. Timed Petri nets and their axiomatization in TRIO

Timed Petri nets [20] differ from traditional Petri nets [25] in that every transition v is associated with a pair of values, usually denoted by $[m_v, M_v]$, belonging to the temporal domain (with $0 \leqslant m_v \leqslant M_v \leqslant \infty$). These are called, respectively, the *lower and upper bound* of v, whereas the pair $[m_v, M_v]$ is called v's *time interval*. Intuitively, the meaning of the pair $[m_v, M_v]$ is that, once v is enabled by the presence of at least one token in each place of its preset, it *cannot fire before a time m_v elapsed* (we call this property LB, since it imposes a lower bound of the firing time of v) and it must fire within M_v unless in the meanwhile it is disabled by the firing of another transition in conflict with it (we refer to this property as UB, since it is related to the upper bound of v). As in traditional Petri nets, tokens are *uniquely* generated and consumed by transition firings. In particular, any firing of a transition consumes one and only one distinct token from each place in its preset (we call this property IU, for input unicity), and introduces one and only one token into each place of its postset; that token can contribute to no more than a single transition firing (we call this property OU, for output unicity).

Formally, a timed Petri net is defined as a 5-tuple: $N = \langle P_N, T_N, F_N, \Theta_N, m_N \rangle$, where

- P_N, T_N, F_N are the set of places, transitions, and arcs of the net; for any transition $v \in T_N$ (resp., place $p \in P_N$) we denote its preset and postset as ${}^\bullet v$ and $v^\bullet$ (resp., ${}^\bullet p$ and $p^\bullet$);
- Θ_N is a function assigning each transition of the net its *time interval*: Θ_N: $T_N \to R_\infty^+ \times R_\infty^+$, where $R_\infty^+ = \{x \mid x \in R \wedge x \geqslant 0\} \cup \{\infty\}$ is the set of non negative reals enriched with the "infinite" value; for each $v \in T_N$, $\Theta_N(v) = \langle m_v, M_v \rangle$ (also denoted as $[m_v, M_v]$ to conform with the literature on the subject) is a pair of nonnegative real values such that $0 \leqslant m_v \leqslant M_v \leqslant \infty$.
- m_N, the initial marking of the net, is a function of type m_N: $P_N \to N$ that assigns to each place of the net its initial marking, i.e. for each place $p \in P_N$ specifies the number $\geqslant 0$ of tokens initially present in it.

In preceding papers ([11] and, in a simplified version, in [12]) we gave a TRIO axiomatization of timed Petri nets, which we briefly report in the following. By means of this axiomatization we can prove, in a sense which will be made precise at the end of the present section, all properties of timed Petri nets executions that can be expressed as TRIO formulas. Since the adopted temporal logic is linear and derivations in axiomatic systems are well suited to prove *valid formulas*, i.e. properties that are satisfied by every model of the axioms, the properties stated and proved in the present approach hold for every possible execution of the net, hence are often called *universal properties*. Universal properties include safety in the traditional sense (e.g., "the system will never enter a failure state", "a marking will always be reached within 2 s from a given initial marking", boundedness, etc). Unavoidable deadlock is also an example of universal property.

For the sake of brevity and simplicity, unless otherwise specified, we adopt the following conventions in writing axioms describing Petri net semantics: all axioms are preceded by an implicit *Alw* operator; identifiers denoting transitions (e.g., r, s, u, v) and places (e.g., p, q) are constant names, while identifiers for time distances (e.g., d, e) are variables; free variables in axioms are implicitly universally quantified at the outermost level. Thanks to the *Generalization* and *Temporal Generalization* meta-theorems the same is true also for the formulas derived from such axioms.

We divide the axioms describing a TPN into general axioms, describing properties that hold for all nets independent of their topology, and topology dependent axioms.

3.1. Basic predicates and general axioms

Since the semantics of timed Petri nets admits multiple simultaneous firings of a single transition, we define a time dependent predicate $nFire(v, n)$, with $n \geqslant 0$, whose meaning is that at the current time transition v fires n times ($n = 0$ iff v does not fire). Of course, the number of transition firings in a given time instant is unique, and there always exists a nonnegative number of firings for each transition: this is expressed, for each transition v, by the two axioms $UFN(v)$ and $NNF(v)$, reported below.

$UFN(v)$ (unique firing number of v):

$$\neg\exists m\exists n(m \neq n \wedge nFire(v, m) \wedge nFire(v, n)),$$

$NNF(v)$ (nonnegative firing of v): $\exists n(n \geqslant 0 \wedge nFire(v, n))$.

To refer individually to the ith firing of a transition v we introduce a time dependent derived predicate $fireth(v, i)$ meaning that there is an ith firing of transition v at the current time; since the intended meaning is that v does fire, we require that $i > 0$. The predicates *fireth* is defined in terms of *nFire* as follows.

DEF(fireth): $fireth(v, i) \stackrel{\text{def}}{=} (i > 0) \wedge \exists n(n \geqslant i \wedge nFire(v, n))$.

To model the timing and topological features of a TPN, we introduce the time-dependent predicate $tokenF(s, i, p, v, j, d)$ meaning that the token produced at the current instant by the ith firing of transition s enters place p and will be consumed by the jth firing of transition v after d time units (this implies $p \in s^{\bullet} \cap {}^{\bullet}v$). *tokenF* is the key predicate in the present axiomatization of TPNs, since each one of its occurrences uniquely identifies a single token produced and consumed in the net. *tokenF* is asserted at the time instant of the firing of the first transition argument, s; for reasons of simplicity and symmetry of the formulas, it is useful to introduce a dual predicate, *tokenP*, having the same meaning but referring to the firing time of the second transition. Predicate *tokenP* is therefore derived from *tokenF* by the following definition:[3]

DEF(*tokenP*): $tokenP(r, i, p, s, j, d) \stackrel{\text{def}}{=} Past(tokenF(r, i, p, s, j, d), d)$.

tokenF implies the firing of the involved transitions, as expressed by the following axiom:[4]

$FI(r, p, s)$ (Future Implication):

$$tokenF(r, i, p, s, j, d) \rightarrow fireth(r, i) \wedge Futr(fireth(s, j), d).$$

3.2. Topology-dependent axioms

These specify, for each transition v of the net, the above informally described properties of upper and lower bound, and of input and output unicity referred to that

[3] From the above definition of *tokenP* the following property, $FP(r, p, s)$: $tokenF(r, i, p, s, j, d) \leftrightarrow Futr(tokenP(r, i, p, s, j, d), d)$ expressed by a formula symmetrical to $DEF(tokenP)$, can be immediately derived.

[4] From $FI(r, p, s)$ and the definition of *tokenP* a symmetrical property, $PI(r, p, s)$ (Past Implication): $tokenP(r, i, p, s, j, d) \leftrightarrow fireth(s, j) \wedge Past(fireth(r, i), d)$ can be immediately derived.

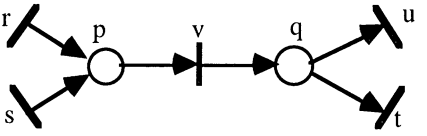


Fig. 1. A simple net F.

transition, in a form that depends on the net topology (i.e., on the places of ${}^\bullet v$ and $v^\bullet$, and on the transitions in their respective presets and postsets). The general form of the topology-dependent axioms is reported in [8]: we illustrate here the axioms $LB(v)$, $UB(v)$, $IU(v)$, and $OU(v)$ specifying such properties for transition v in the simple net fragment F of Fig. 1, to give the reader an intuitive understanding of how they can be expressed as TRIO axioms.

$$LB(v): \quad fireth(v,i) \rightarrow \exists d(d \geqslant m_v \wedge \exists j(tokenP(r,j,p,v,i,d) \vee tokenP(s,j,p,v,i,d))),$$

$$UB(v): \quad (fireth(r,i) \rightarrow \exists d(d \leqslant M_v \wedge \exists j\, tokenF(r,i,p,v,j,d)))$$
$$\wedge\, (fireth(s,i) \rightarrow \exists d(d \leqslant M_v \wedge \exists j\, tokenF(s,i,p,v,j,d))).$$

$$IU(v): \quad tokenP(x,i,p,v,j,d) \wedge tokenP(y,k,p,v,j,e) \rightarrow x = y \wedge i = k \wedge d = e$$

(with x and y variables ranging on the set of transitions),

$$\text{OU(v)}: \quad tokenF(v,i,q,x,j,d) \wedge tokenF(v,i,q,y,k,e) \rightarrow x = y \wedge j = k \wedge d = e$$

(with x and y variables ranging on the set of transitions).

Unlike traditional Petri nets, the instantaneous marking of a TPN does not adequately characterize its overall state: the "age" of each token (i.e., the length of the time interval elapsed from the time of its creation to the present time) is significant too. To uniquely identify tokens, we refer to the events of their production and consumption (i.e., to the firings of transitions), and to the places where they are inserted, or from which they are deleted. The temporal semantics of the nets is ultimately provided by imposing constraints on the distance in time among transition firings. The notion of *instantaneous marking* is formalized in [11, 12] as a *derived concept* on the basis of transition firing axioms. We do not report such formalization here because in the present work we focus our attention on the transition firings, which in our approach constitute the observable events of interest.

We however formalize the initial marking of the net, because it is an essential part of the net definition. Let us assume that in the initial state of the net all the tokens have just been created, so that their age is zero: this is the most frequently assumption adopted in the literature and, as it will be apparent from the following, generalizations under this respect are straightforward. As shown in Fig. 2, for each place p in the net, we introduce a special extra transition called *itp* (initializing transition for p) with $itp^\bullet = \{p\}$, ${}^\bullet itp = \emptyset$. If p's initial marking is k, then the following axiom holds (notice

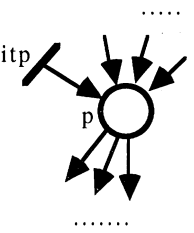


Fig. 2. The dummy transition that "builds" the initial marking.

that no *Alw* operator is implicitly assumed in this axiom):

$$\mathrm{IM}(p) \quad nFire(itp, k) \wedge AlwF(nFire(itp, 0)) \wedge \forall x\ AlwP(nFire(x, 0)).$$

where variable x ranges over the set T_N of the transitions of net N (including *itp*). Axiom $IM(p)$ states that *itp* only fires k times at the current instant (and never before or after now) and that no transition ever fired before.

In summary, given a timed Petri net N, its temporal features are characterized by the set of axioms

$$Ax(N) = Axg(N) \cup Axt(N)$$

where $Axg(N)$ and $Axt(N)$ are, respectively, the sets of general and topological axioms of the net. These, in turn, are composed as follows:

$$Axg(N) = UFN_N \cup NNF_N \cup FI_N,$$

where

$$UFN_N = \{UFN(t) \mid t \in T_N\},$$

$$NNF_N = \{NNF(t) \mid t \in T_N\},$$

$$FI_N = \{FI(r, p, s) \mid r, s \in T_N, p \in r^\bullet \cap {}^\bullet s\}$$

and $\quad Axt(N) = LB_N \cup UB_N \cup IU_N \cup OU_N \cup IM_N,$

where

$$LB_N = \{LB(t) \mid t \in T_N\},$$

$$UB_N = \{UB(t) \mid t \in T_N\},$$

$$IU_N = \{IU(t) \mid t \in T_N\},$$

$$OU_N = \{OU(t) \mid t \in T_N\},$$

$$IM_N = \{IM(t) \mid t \in T_N\}.$$

Therefore, for each given timed Petri net N a theory is uniquely determined, whose proper axioms are precisely $Ax(N)$. In ther words, there is a correspondence *th*

between TPNs and their theories. For the sake of conciseness, in the rest of the paper, for a given timed Petri net N we will denote the theory $th(N)$ as $\mathcal{N}$.

Let us consider the TRIO formulas that can be constructed starting from the following alphabet:
– constants or variables representing places and transitions,
– (real-valued) constants or variables representing distances among time instants or length of time intervals,
– the usual arithmetic operators ' + ', ' − ', '*', '/', … on terms representing time,
– the equality predicate " = " for time-, place-, or transition-valued terms, the less-then predicate ' < ' for time-valued terms,
– the predicates *nFire* and *tokenF*.
For any TRIO formula φ constructed as specified above, if $\vdash_{\mathcal{N}} \varphi$ then every execution of net N satisfies the property described by φ.

As an example of a timing property consider, with reference to the net fragment F of Fig. 1, the formula

$$fireth(r, i) \rightarrow WithinF(\exists j(fireth(u, j) \oplus fireth(t, j)), \quad M_v + \max(M_u, M_t))$$

(where the propositional connective '$\oplus$' denotes exclusive disjunction) which asserts that any firing of transition r will always be followed by a corresponding firing of either t or u within $M_v + \max(M_u, M_t)\, t.u.$ It can be derived, in the theory $\mathcal{F}$ of net F, using $UB(v)$, $UB(u)$, $UB(t)$, $FI(v, q, t)$, $FI(v, q, u)$, and the definition of $WithinF$.

4. Implementation relation among TPNs

In this section we characterize the notion of implementation relation among TPNs. An implementation must satisfy, by its very definition, all the requirements expressed in the specification, hence we say that a timed Petri net I implements a timed Petri net S acting as a specification if all the properties satisfied by net S are also ensured by net I. To make this precise we must therefore provide the following items of information: (1) what kind of properties of TPNs we require to preserve implementations? (2) how these properties are to be ensured by the implementation net (in other words, if net S has property π, which is the property ψ that must hold in I?); and (3) the precise conditions, according to the above items, under which two given nets S and I are in the implementation relation. We answer these questions by an informal description in the paragraphs below, and provide formal definitions subsequently.

(1) In our view the properties ensured by a timed Petri net are the temporal relations among transition firings that are verified in *every* execution of the net. We therefore assume that transition firings are the only observable events of the net: the state of a TPN, as usually defined in terms of place marking and age of the tokens, is assumed to be an internal feature, not accessible to the observer. Properties of this kind are naturally described by means of TRIO formulas. These assumptions are formally stated in Definition 1 (Observable property) below.

(2) An obvious notion of implementation would require that the properties to be satisfied by net I are exactly the same that are ensured by S; this implies that every transition of net S corresponds to exactly one transition of I that "implements" it. Although this is a plausible choice, we consider it too restrictive, since we would like to consider also the case, frequently encountered in practice, where a single action a in the high-level system description is implemented by several, actions $a_1, \ldots, a_n$, that in the low-level description can occur in a mutually exclusive fashion depending on some condition that for abstraction purposes is ignored in the high-level version. Therefore, in our approach every transition t of net S corresponds, in net I, to a set of transitions $t_1, \ldots, t_n$, $n \geqslant 1$, as specified in Definition 2 (Event function); furthermore, if net S satisfies a formula asserting that t fires, then in net I a formula must hold, asserting that one of $t_1, \ldots, t_n$ fires, as stated in Definition 3 (Property function).

(3) The precise definition of the implementation relation follows directly from the above items: nets S and I are in the implementation relation if the formulas describing properties of S as in (1) above, translated as described in (2) above, are satisfied by net I. This is precisely what stated in Definition 4 (implementation relation),

Next we define formally the formulas describing an observable property of a TPN, the correspondence among transitions in TPNs associated in the implementation relation, and the implementation relation itself.

Definition 1 (*Observable formula and property*). Given a timed Petri net N, an observable formula for N is a TRIO formula constructed on the time-dependent predicate *nFire* (and on the derived predicate *fireth*) applied to transitions in T_N, plus the usual arithmetic predicates and functions on the temporal domain. An observable *property* φ is an observable formula that can be derived as a theorem for theory $\mathcal{N}$, i.e., such that $\vdash_{\mathcal{N}} \varphi$ holds.

The definition of observable property refers only to the *nFire* predicate because we assume that transition firings are the only observable events in the net. The other fundamental predicate in the axiomatization of timed Petri nets, namely predicate *tokenF*, is related to the topology of the net and models the cause-effect relation among transition firings: such information concerns the mechanism of token production and consumption, so we consider it immaterial for the implementation relation.

Definition 2 (*Event function*). Given two TPNs S and I, an event function from I to S is any onto function $\lambda: T_I \to T_S$ from the transition of I to the transitions of S.

An event function from net I to net S specifies which transition of I represents transitions of S in a (possible) implementation relation. Notice that λ may be partial, since the net I may add details (transitions) that are not present in S, but it is required to be onto, because *every* (universal) property of S must be ensured by I, which implies that every transition of S is represented by some transition in I; furthermore, λ is not required to be one to one: a single transition of S may well be represented by more

than one transition of I (e.g., when an action of S is implemented by two exclusive actions of I).

Definition 3 (*Property function*). Given two TPNs S and I and an event function λ from I to S, a property function Λ (uniquely determined by λ) from $\mathscr{S}$ to $\mathscr{I}$ is a function that translates observable formulas of S into observable formulas of I, according to the following requirements[5]

(i) $\Lambda(nFire(v, n)) = \exists n_1 \ldots \exists n_s(n_1 + \cdots + n_s = n \wedge nFire(v_1, n_1) \wedge \cdots \wedge nFire(v_s, n_s))$ where $\{v_1, \ldots v_s\} = \{t \in T_I | \lambda(t) = v\}$ is the set of transitions net I into which v is refined;
(ii) $\Lambda(P(t_1 \ldots t_n)) = P(t_1 \ldots t_n)$ for any other atomic formula (i.e., when P is '$=$' or '$<$');
(iii) $\Lambda(\alpha \to \beta) = \Lambda(\alpha) \to \Lambda(\beta)$;
(iv) $\Lambda(\mathrm{Dist}(\alpha, t)) = \mathrm{Dist}(\Lambda(\alpha), t)$;
(v) $\Lambda(\neg \alpha) = \neg \Lambda(\alpha)$;
(vi) $\Lambda(\forall x \alpha) = \forall x \Lambda(\alpha)$.

Remark (On function Λ). Notice that from clause (i) of Definition 3 above, it follows that

$$\Lambda(nFire(v, 0)) = nFire(v_1, 0) \wedge \cdots \wedge nFire(v_s, 0) \text{ if } \{v_1, v_s\} = \{t \in T_I | \lambda(t) = v\};$$

furthermore, if $\{v_1\} = \{t \in T_I | \lambda(t) = v\}$, that is, v_1 is the only transition in net I corresponding to transition v in net S, then

$$\Lambda(nFire(v, k)) = nFire(v_1, k) \text{ for any integer number } k.$$

The above properties simplify the translation of formulas via Λ in case a transition does not fire or the correspondence among transitions is one-to-one.

The above introduced notations allow us to formally characterize the implementation relation among TPNs.

Definition 4 (*Implementation relation among TPNs*). Given two TPNs S and I and an event function λ from I to S (and accordingly a property function Λ from $\mathscr{S}$ to $\mathscr{I}$), we say that I implements S *through* λ iff, for each observable formula φ of S, $\vdash_{\mathscr{S}} \varphi$ implies $\vdash_{\mathscr{I}} \Lambda(\varphi)$, i.e., the translation by Λ of every observable property of S is an observable property of I. We say that I implements S iff there exists an event function λ from I to S such that I implements S through λ.

[5] Since any TPN has only a finite set of transitions, we assume, without loss of generality, that only transition constants appear in observable formulas; if a formula contains a variable representing a transition (necessarily quantified, since we consider only sentences), the quantification can be translated into a finite conjunction or disjunction of formulas where only transition constants occur.

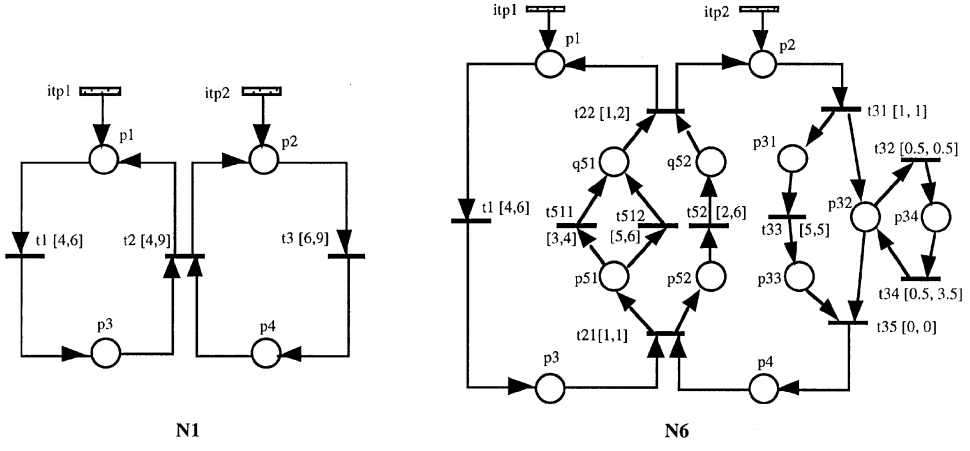


Fig. 3. A simple TPN (N_1) and a more complex net (N_6) implementing it.

It can be easily shown that the implementation relation as defined above is transitive: given nets N_1, N_2, and N_3, if N_2 implements N_1 through event function λ_1 and N_3 implements N_2 through function λ_2, then N_3 implements N_1 through the event function $\lambda_1 \circ \lambda_2$. This property can be the base for a design methodology where the implementation is obtained in terms of a sequence of *refinement* steps.

Example 1. As a trivial example of implementation relation among timed Petri nets, let us consider two TPNs S and I having the same topology and initial marking (i.e., $T_S = T_I$, $P_S = P_I$, $F_S = F_I$, and $m_S = m_I$) and such that the time interval of every transition in I is stricter than that of the same transition in S (i.e., $\forall v \in T_I$, if we let $\Theta_I(v) = [m_{vI}, M_{vI}]$ and $\Theta_S(v) = [m_{vS}, M_{vS}]$ then $m_{vI} \geqslant m_{vS}$ and $M_{vI} \leqslant M_{vS}$). Then it can be easily proven that net I implements net S. In fact, every axiom $\alpha_S \in Ax(S)$ for net S is logically implied by the corresponding axiom $\alpha_I \in Ax(I)$ in I, i.e., $\vdash \alpha_I \rightarrow \alpha_S$, because of the stricter time bounds. Then net I implements net S through the identity event function, so that Λ is the identity function on the observable formulas of net S. In fact, for any observable formula φ, $\vdash_{\mathscr{S}} \varphi$ implies $\vdash_{\mathscr{I}} \varphi$ thanks to the following property of the TRIO axiomatic calculus (and in fact of any first order calculus): if $\Gamma; \alpha \vdash \varphi$ and $\vdash \beta \rightarrow \alpha$ then $\Gamma; \beta \vdash \varphi$.

Example 2. As a more concrete example[6] of implementation, consider the net fragments N_1 and N_6 shown in Fig. 3. N_1 models a simple rendezvous between a producer and a consumer. The producer gets data (e.g., temperature, pressure) from an external device, in 4–6 time units (transition t_1), and then, in 4–9 t.u., communicates the

[6] The example is borrowed, with modifications, from [FGP93].

acquired data to the consumer who is responsible for the elaboration (transition t_2). The elaboration by the consumer takes 6–9 t.u. (transition t_3). Initially, the producer is ready for the acquisition and the consumer is ready for elaboration (places p_1 and p_2 are initially marked: notice initializing transitions itp_1 and itp_2). In the next section we will prove that the net N_6, where several elements modeling design choices have been added to those of N_1, implements it, with event function λ defined as $\lambda(t_1) = t_1$, $\lambda(t_{22}) = t_2$, $\lambda(t_{35}) = t_3$, and $\lambda(t) = \perp$ for every other transition $t \in T_{N_6}$.

4.1. A method for proving implementation

Proving the existence of an implementation relation among two arbitrary TPNs can be complex and difficult. First of all, the choice of the correct event function may be nontrivial, since the space of such function is rather large. Furthermore, given an event function, proving the implementation relation requires the proof of a meta-theorem on the derivability of a set of formulas, and no precise guideline on how to structure such proof can be given if no additional information is available on the relation among the two given nets. This situation is further complicated by the fact that (timed) Petri nets are rather unstructured mathematical objects: in general they are just bipartite graphs having no a priori constraint on the adjacency relation among nodes.

In the following we present a method for proving implementation among TPNs that can be easily applied when the implementation mechanism is intuitively clear, as it is in the case of an implementation net obtained through systematic transformation of a given specification net. The method is based on the idea that for each observable property π of specification net S there exists in the axiomatization of the implementation net I a proof of $\Lambda(\pi)$ that mirrors the proof of π in $Ax(S)$. We therefore introduce a *proof* function Δ that translates formulas of theory $\mathscr{S}$ (referring to the transitions of S) included in the derivation of $\vdash_{\mathscr{S}} \pi$ into formulas of theory $\mathscr{I}$ (referring to transitions of I) in such a way that if the derivation of $\vdash_{\mathscr{S}} \pi$ consists of the formulas $\pi_0, \pi_1, \ldots, \pi_n$ (where $\pi_n = \pi$), then the proof of $\vdash_{\mathscr{I}} \Lambda(\pi)$ includes formulas $\Delta(\pi_0), \ldots, \Delta(\pi_1), \ldots, \Delta(\pi_n) = \Delta(\pi)$ plus possibly other ones. The proof of π for net S uses axioms of $Ax(S)$ that describe the cause–effect relations among transition firings and therefore include occurrences of the *tokenF* predicate. The proof translation function must therefore be defined on any formula of theory $\mathscr{S}$, not only on observable formulas. Finally, notice that the requirements expressed above for the proof translation function Δ imply that $\Delta\gamma(\pi) = \Lambda(\pi)$, i.e., Δ must be an extension of the property function Λ. As it will be apparent in the following, Δ is therefore essentially characterized by the way it translates the *tokenF* predicate.

The systematic translation of the derivation of $\vdash_{\mathscr{S}} \varphi$ into the derivation of $\vdash_{\mathscr{I}} \Lambda(\varphi)$ is formalized by the notion of *proof* (translation) function, to be defined next.

Definition 5 (*Proof function*). Given two TPNs S and I, with functions λ and Λ as in Definition 3, a proof (translation) function Δ from $\mathscr{S}$ to $\mathscr{I}$ is a function that translates

any well-formed formula (wff) of S into a wff of I satisfying the following conditions.
(i) Δ is an extension of Λ (i.e., it is equal to Λ where they are both defined), and
(ii) Δ is compositional with respect to the structure of the following formulas:

$$\Delta(\alpha \to \beta) = \Delta(\alpha) \to \Delta(\beta); \qquad \Delta(\neg \alpha) = \neg \Delta(\alpha);$$

$$\Delta(Dist(\alpha, t)) = Dist(\Delta(\alpha), t); \qquad \Delta(\forall x \alpha) = \forall x \Delta(\alpha).$$

The proof function is extended to sets of formulas in an obvious way: for a set Γ of wffs of theory $\mathscr{S}$, we define $\Delta(\Gamma)$ as the set $\{\Delta(\gamma) | \gamma \in \Gamma\}$ including precisely the translation according to Δ of all wffs of Γ.

The following proposition illustrates some (immediately proved) properties of the proof function.

Proposition 6. *If Δ is any proof function from wffs of a TRIO theory $\mathscr{S}$ to wffs of a TRIO theory $\mathscr{I}$*, then
(a) *Δ is congruous with modus ponens*, i.e., $\{\Delta(\alpha), \Delta(\alpha \to \beta)\} \vdash_{\mathscr{I}} \Delta(\beta)$;
(b) *TRIO axioms are maintained by Δ, i.e., the translation of each TRIO axiom of theory $\mathscr{S}$ is a theorem of theory $\mathscr{I}$.*

The following metatheorem provides the basis for our principal method of proving implementation among TPNs.

Metatheorem 7. *Let S and I be two TPNs, $\mathscr{S}$ and $\mathscr{I}$ the two TRIO theories describing them, Δ a proof function from $\mathscr{S}$ to $\mathscr{I}$. Then for every wff φ of $\mathscr{S}$, $\vdash_{\mathscr{S}} \varphi$ implies $\Delta(Ax(S)) \vdash_{\mathscr{I}} \Delta(\varphi)$.*

The proof of the metatheorem is by induction on the length of the derivation of φ in $\mathscr{S}$. Let $\varphi_0, \varphi_1, \ldots, \varphi_n = \varphi$ be a derivation of φ in $\mathscr{S}$; then for each i, with $0 \leqslant i \leqslant n$, $\Delta(Ax(S)) \vdash_{\mathscr{I}} \Delta(\varphi_i)$.

Base step: (1) If φ_0 is a TRIO axiom then $\vdash_{\mathscr{I}} \Delta(\varphi_0)$ by Proposition 6(b);

(2) If $\varphi_0 \in Ax(S)$, then $\Delta(\varphi_0) \in \Delta(Ax(S))$, hence $\Delta(Ax(S)) \vdash_{\mathscr{I}} \Delta(\varphi_0)$.

Induction step: Let us assume that $\Delta(Ax(S)) \vdash_{\mathscr{I}} \Delta(\varphi_j)$ for each $j < i$; then

(1) if $\varphi_i \in Ax(S)$ or φ_i is a TRIO axiom, then $\Delta(Ax(S)) \vdash_{\mathscr{I}} \Delta(\varphi_i)$ as in the base case;

(2) if φ_i is obtained by modus ponens from two preceding formulas φ_h and $\varphi_k = \varphi_h \to \varphi_i$, with $h, k < i$, then by the induction hypothesis $\Delta(Ax(S)) \vdash_{\mathscr{I}} \Delta(\varphi_k)$ and $\Delta(Ax(S)) \vdash_{\mathscr{I}} \Delta(\varphi_h \to \varphi_i)$ and the thesis follows from Proposition 6(a). □

The preceding metatheorem shows that a proof translation function provides, as its name suggests, a way to obtain, from a proof of φ in $\mathscr{S}$, a proof of $\Delta(\varphi)$ in $\mathscr{I}$ from $\Delta(Ax(S))$.

Given two sets of wffs Γ and Ψ, with a slight abuse of notation we shall write in the following $\Gamma \vdash \Delta(\Psi)$ meaning that $\Gamma \vdash \Delta(\psi)$ for each $\psi \in \Psi$ or, equivalently, that $\Gamma \vdash \bigwedge_{\psi \in \Psi} \Delta(\psi)$.

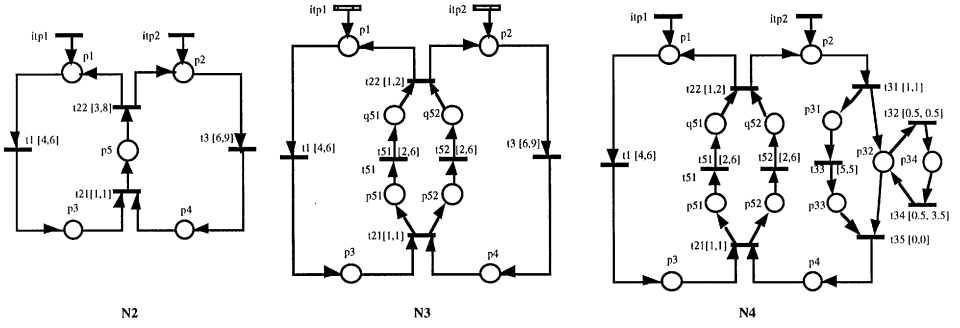


Fig. 4. A sequence of refinement steps.

Corollary 8 (To Metatheorem 7). *In the same hypotheses as in Metatheorem* 7, *if* $\vdash_{\mathscr{I}} \Delta(Ax(S))$ (*i.e., all axioms of* $\mathscr{S}$, *translated into* $\mathscr{I}$, *are theorems*) *then net I implements S* (*i.e., for each observable formula* φ, $\vdash_{\mathscr{S}} \varphi$ *implies* $\vdash_{\mathscr{I}} \Lambda(\varphi)$).

Corollary 8 provides a sufficient condition for implementation: to prove that a TPN I implements a TPN S through event function λ, it suffices to find a proof function Δ such that the axioms of net S, translated through Δ, are theorems of the theory $\mathscr{I}$ of TPN I.

Example 3. We apply Corollary 8 to the TPNs N_1 of Fig. 3 and N_2 of Fig. 4; N_2 implements N_1 through event function defined as follows: $\lambda(t_{22}) = t_2$, $t_{21} \notin \mathrm{Dom}(\lambda)$, and $\lambda(x) = x\ \forall x \in T_{N_2} - \{t_{21}, t_{22}\}$. Implementation can be proved by taking Λ defined in the obvious way, and Δ extending Λ as follows: for all i, j, d,

$$\Delta(tokenF(t_1, i, p_3, t_2, j, d)) = \exists d_1 \exists d_2 \left(d_1 + d_2 = d \wedge \exists k \left(\begin{array}{l} tokenF(t_1, i, p_3, t_{21}, k, d_1) \\ \wedge \\ Futr(tokenF(t_{21}, k, p_5, t_{22}, j, d_2), d_1) \end{array} \right) \right)$$

$\Delta(tokenF(t_3,\ i, p_4, t_2, j, d))$ is defined similarly, and $\Delta(tokenF(u, i, p, v, j, d)) = tokenF(\lambda^{-1}(u), i, p, \lambda^{-1}(v), j, d)$ for any other pair of transitions u and v.[7]

Intuitively, the proof function describes how any execution of net N_1 can be simulated by an execution of net N_2. In net N_1 a token produced by a firing of transition t_1 (or t_3) can be consumed d time units later by a firing of transition t_2, then in net N_2 a token produced by a firing of t_1 (or t_3) can be consumed by a firing of t_{21} occurring d_1 after time units and this firing produces a token that can be consumed by

[7] Notice that in this case λ^{-1} is defined because λ is one-to-one.

a firing of t_{22} d_2 time units later, with $d_1 + d_2 = d$. All other events of token production and consumption taking place in net N_1 can be simulated directly by net N_2.

The adequacy of the above-defined proof function to prove that N_2 implements N_1 will be discussed, in a more general setting, in Section 5.1.1.

Remark (*On initial marking*). The above-described method for proving implementation can consider also the initial marking of the TPNs, despite the fact that it refers exclusively to the transitions and their firings. The reason for this is precisely the fact that the initial marking of places is defined, in Section 3.2, in terms of suitable firings of special initializing transitions. In general, if place p in net S is initially marked, then an axiom of $\mathscr{S}$ called $IM(p)$ describes the firings of the initializing transition itp_s. If a second net I implements S then it must have a transition ipt_i, corresponding to itp_s, which fires in such a way that $\Lambda(IM(p))$ is a theorem of $\mathscr{I}$ As a concrete case, consider again nets N_1 and N_2 of Example 9. In net N_1, place p_1 is initially marked with one token, therefore the following sentence

$$IM(p_1) = nFire(itp_1, 1) \wedge AlwF(nFire(itp_1, 0)) \wedge \forall x\, AlwP(nFire(x, 0))$$

is part of the axiomatization of N_1. Also net N_2 has place p_1 marked, which is modeled by an initializing transition, also called itp_1, with an axiom $IM(p_1)$ identical to that of N_1. Now since $\lambda(itp_1) = itp_1$ and the correspondence between the two transitions is one-to-one, $\Lambda(IM(p_1)) = IM(p_1)$ and the translated axiom is obviously a theorem for the net N_2.

5. Implementation through refinement

An incremental approach to the specification and design of time critical systems through timed Petri nets and TRIO can avoid many of the difficulties in proving implementation outlined in Section 4.1 and in general it can greatly reduce the overall development effort. In the following we present a set of *refinement rules* that, given a TPN, permit the substitution of one of its components, be it a transition or a place, with a net fragment composed of a combination of new places and transitions in such a way that the new net retains the properties of the initial net and is therefore an implementation. The correctness of each of these refinement rules can be proved in a general way, independently of particular net instances, under the hypothesis that the firing time intervals of the newly introduced transitions satisfy suitable constraints with respect to those of the transitions composing the original net.

An incomplete sample of such refinement rules is represented in Tables 2 and 3. Each table reports in the first column the net fragment where the substitution is performed and in the other columns the net resulting from the rule application. For each rule, the table also provides timing constraints (TC) among the time bounds of

Table 2
Transition refinement rules

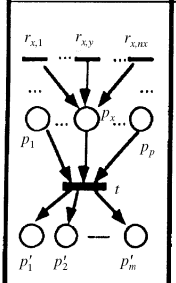
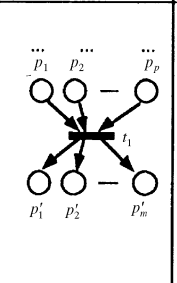
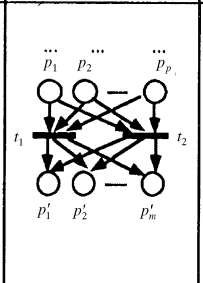
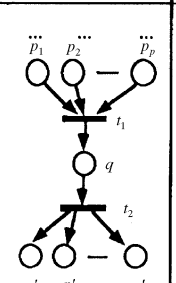
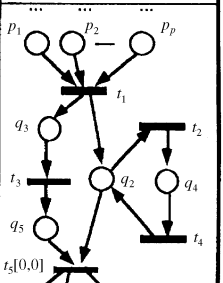
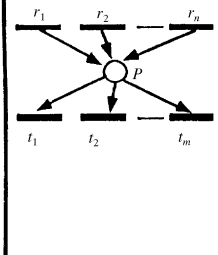
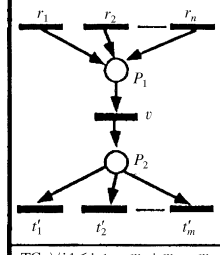
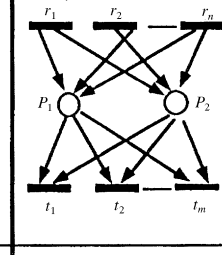
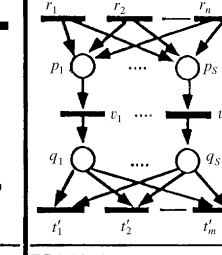
(a)	(b) Firing times restriction	(c) Transition splitting	(d) Transition sequencing	(e) Iteration
				
	TC: $m_t \leqslant m_{t_1} \leqslant M_{t_1} \leqslant M_t$ EF: $\lambda(t_1) = t$	TC: $m_{t_1} = m_{t_2} = m_t$ $M_{t_1} = M_{t_2} = M_t$ EF: $\lambda(t_1) = t$ $\lambda(t_2) = t$	TC: $m_{t_1} + m_{t_2} = m_t$ $M_{t_1} + M_{t_2} = M_t$ EF: $t_1 \notin Dom(\lambda)$ $\lambda(t_2) = t$	TC: $0 < m_{t_2} = M_{t_2} = ex$ $m_{t_3} = M_{t_3} = dx$ $M_{t_1} + M_{t_2} + M_{t_3} + M_{t_4} = M_t$ $m_{t_1} + m_{t_3} = m_t$ EF: $t_1, t_2, t_3, t_4 \notin Dom(\lambda)$ $\lambda(t_5) = t$

Table 3
Place refinement rules

(a)	(b) Place sequencing	(c) Place splitting	(d) Process splitting
			
	TC: $\forall i\, 1 \leqslant i \leqslant m \quad m_v + m_{t_i'} = m_{t_i}$ $M_v + M_{t_i'} = M_{t_i}$ EF: $v \notin Dom(\lambda)$ $\forall i\, 1 < i \leqslant m \quad \lambda(t_i') = t_i$ $\forall i\, 1 < i \leqslant n \quad \lambda(r_i') = r_i$	EF: $t_1 \notin Dom(\lambda)$ $\forall i\, 1 < i \leqslant m \quad \lambda(t_i) = t_i$ $\forall i\, 1 < i \leqslant n \quad \lambda(r_i) = r_i$	TC: $\forall i\, 1 \leqslant i \leqslant m \quad m_v + m_{t_i'} = m_{t_i}$ $M_v + M_{t_i'} = M_{t_i}$ $\forall i\, 1 \leqslant i \leqslant s \quad m_{v_i} = m_v \quad M_{v_i} = M_v$ EF: $v_i \notin Dom(\lambda) \quad \forall i\, 1 < i \leqslant m \quad \lambda(t_i') = t_i$ $\forall i\, 1 < i \leqslant n \quad \lambda(r_i') = r_i$

the involved transitions, and the event function (EF) for such transitions (since the rest of the net is unchanged, $\lambda(v) = v$ for every other transition).

Table 2 displays *transition* refinement rules, where a single transition t in net S is replaced in net I by a net fragment; for all such rules (except the transition splitting rule of column (c)) it is required that in net S there is no transition conflicting with t. Table 3 displays *place* refinement rules, where the refined component in net S is a place p; for all such rules (except the process splitting rule of column (c)) it is required that no transitions $t_1, \ldots, t_m$ of $p^\bullet$ is in the postset of any other place.

Table 2(d) describes the *transition sequencing* rule that substitutes a given transition t with a sequence of two transitions t_1 and t_2 and a place p among them representing the start and the end of the action modeled by the original transition t. We apply this rule to transition t_2 of net N_1 in Fig. 3 obtaining the net N_2 of Fig. 4. The action representing the communication between the producer and the consumer is detailed in two actions (transitions t_{21} and t_{22}) representing the start and the end of the communication, respectively. N_2 implements N_1 through the event function λ_1:

$$\lambda_1(t_1) = t_1; \qquad \lambda_1(t_3) = t_3; \qquad \lambda_1(t_{22}) = t_2, \qquad \lambda_1(t_{21}) = \perp.$$

Table 3(d) presents the *process splitting* rule: a place p and the transitions $\{t_1, \ldots, t_m\}$ in its postset are replaced by a set of transitions $v_1, \ldots, v_s$ having $p_1, \ldots, p_s$ as preset places and $q_1, \ldots, q_s$ as postset places, with each of the q_i's being in the preset of every transition of $\{t'_1, \ldots, t'_m\}$. Intuitively, the application of this rule introduces a set of s processes that evolve in parallel, being synchronized at the start and at the end of their evolution. This rule, applied to N_2 in Fig. 4, yields the net N_3. Place p_5, representing the communication being held, is refined into two processes (p_{51} and p_{52} with transitions t_{51} and t_{52}) in net N_3 to model the actions executed in parallel during the communication by the producer and the consumer. N_3 implements N_2 through event function λ_2:

$$\lambda_2(tx) = tx \ \ \forall tx \in \{t_1, t_{22}, t_3, t_{21}\}, \quad \lambda_2(t_{51}) = \lambda_2(t_{52}) = \perp.$$

Table 3(b) shows the *place sequencing* rule: a place p and the transitions $\{t_1, \ldots, t_m\}$ in its postset are replaced by places p_1 and p_2, transition v and a set of transitions $\{t'_i\}$ each corresponding to one of the transitions $t_i \in p^\bullet$.

Table 3(c) shows the *place splitting* rule: a place p is replaced by places p_1 and p_2, with the same preset and postset as p.

Table 2(e) presents the *iteration* rule. It consists of substituting a transition t with a set of transition and places modeling repeated firings of a transition with a time upper bound equal to that of the original transition t. Applying this rule to transition t_3 of N_3 of Fig. 4 leads to the net N_4. The consumer's behavior is described by two processes: an iteration executing the computation, controlled by a time-out. Now the event function is

$$\lambda_3(t_{35}) = t_3; \quad \lambda_3(t_x) = tx \ \ \forall tx \in \{t_1, t_{22}, t_{51}, t_{21}\},$$

$$\lambda_3(t_{31}) = \lambda_3(t_{32}) = \lambda_3(t_{33}) = \lambda_3(t_{34}) = \perp.$$

Table 2(c) presents the *transition splitting* rule, where a transition t is split into two transitions t_1 and t_2 having the same preset and postset and the same firing time interval as t. This rule can be applied to model that the original action is implemented by two alternatives. The net N_5, obtained by applying this rule to transition t_{51} in the net of N_4 of Figure 4, has the same topology as the net N_6 in Fig. 3, with different time intervals for transitions t_{511} and t_{512}, which are associated with the interval [2,6] as the original transition t_{51}. Transition t_{51} of net N_4, representing actions performed by

the producer, can be further refined into two transitions (t_{511}, t_{512}) representing two exclusive actions (e.g., an "if" inside the code of the producer). The net N_5 implements N_4 with the event function:

$$\lambda_4(t_{511}) = \lambda_4(t_{512}) = t_{51}; \quad \lambda_4(tx) = tx \text{ for the other transitions.}$$

Table 2(a) describes, finally, the *firing times restriction* rule: a given transition t is substituted by a new transition t_1 having a restricted firing time interval. This rule formalizes the concepts presented in Example 1. Applying this rule to the transitions t_{511} and t_{512} just obtained yields the net N_6 of Fig. 2, where t_{511} and t_{512} represent actions having different timings. N_6 implements N_5 with the identity event function: $\lambda_5(tx) = tx$.

In conclusion, the net N_6 implements the net N_1 through the event function $\lambda = \lambda_5 \circ \lambda_4 \circ \lambda_3 \circ \lambda_2 \circ \lambda_1$.

5.1. Proving correctness of refinement rules

In this section we show how Corollary 8, which suggests a method for proving the implementation among TPNs, can also be used for the proof of correctness of the refinement rules.

To provide such proof, we must show that every net containing a fragment like that of Tables 2(a) or 3(a) is correctly implemented by any net having the same places and transitions, except that the highlighted fragment is substituted by one of the fragments in Table 2(b)–2(e) or 3(b)–3(d). From this point on we call the net containing fragment *a* also net *S*, because it acts as specification net, whereas nets containing fragments *b..e* will be called net *I*, because they are implementation nets.

First, we define for every rule a different *proof* translation function Δ, that translates every formula for the specification net *S* into a formula for the refined net *I*. Function Δ must be an extension of the *property* function Λ. Λ is characterized in Definition 3 by the *event* function λ described in the tables. The proof function Δ must be defined on every predicate of net *S*, included *tokenF*; however, since Δ is required to be extension of Λ, it is essentially characterized by the way it translates *tokenF* predicate.

In the following, for each refinement rule, we propose one possible definition (the more intuitive one) of the proof function Δ and we explain briefly the intuitive meaning of the translation. The functions thus defined are adequate to prove the correctness of refinement rules. According to Corollary 8, to prove that the refined net is an implementation, one must show that the axioms of net *S*, translated through Δ, are theorems of theory $\mathscr{I}$, i.e. $\vdash_{\mathscr{I}} \Delta(Ax(S))$.

As explained in Section 3.2,

$$Ax(S) = Axg(S) \cup Axt(S)$$

$$= UFN_S \cup NNF_S \cup FI_S \cup LB_S \cup UB_S \cup IU_S \cup OU_S \cup IM_S.$$

In principle, therefore, the proof that $\vdash_{\mathscr{I}} \Delta(Ax(S))$ consists of a number of proofs that $\vdash_{\mathscr{I}} \Delta(\sigma)$, for each $\sigma \in Ax(S)$. The proof functions are however defined in such a way that $\Delta(\sigma) = \sigma$ for any axiom σ pertaining only to transitions that are not involved in the application of the refinement rule, and furthermore for such axioms $\sigma \in Ax(I)$, because σ is related to a portion of the net S that remains, unchanged, in net I. For this kind of axiom, therefore, there is really nothing to prove. The significant part of the proof regards the axioms connected with transitions involved in the refinement. Of these axioms, the proofs for those belonging to $Axg(S)$ are trivial, while for the topological axioms $Axt(S)$ the proof may be straightforward and very intuitive, or difficult and intricate, depending on the topology of the studied fragments. For brevity, in the present section we only outline one significant proof for each refinement rule. Complete detailed correctness proofs are reported in [8].

Notice that for simplicity in Table 2 we drew only the transitions in the preset of each of the places $p_1 \ldots p_p$ that are in the preset of transition t. We will assume that each place p_x, with $x \in [1..p]$, has n_x transitions in its preset. In the following of Section 5.1 we will call $r_{x,y}$, with $x \in [1..p]$ and, for each x, for $y \in [1..n_x]$ the yth transition in the preset of the xth place in the preset of transition t.

5.1.1. Firing times restriction rule

In this case the Δ function is very simple.

$$\Delta(tokenF(u, i, p, v, j, d)) = tokenF(\lambda^{-1}(u), i, p, \lambda^{-1}(v), j, d) = tokenF(u, i, p, v, j, d)$$
if $u \neq r_{x,y}$ or $v \neq t$;

$$\Delta(tokenF(r_{x,y}, i, p_x, t, j, d)) = tokenF(r_{x,y}, i, p_x, t_1, j, d)$$

A token produced by $r_{x,y}$ and consumed by t is translated into a token produced again by $r_{x,y}$ and consumed by t_1.

To prove correctness of this refinement rule, we simply observe that the axioms of net S, translated through Δ, are unchanged, except for the fact that transition t_1 replaces transition t. Besides, it is a trivial remark that every sentence obtained through the translation of an axiom for net S is logically implied by the corresponding axiom in I (see also example 1).

5.1.2. Transition sequencing rule

The proof translation function applied to the *tokenF* predicate is defined as follows.

$$\Delta(tokenF(u, i, p, v, j, d)) = tokenF(\lambda^{-1}(u), i, p, \lambda^{-1}(v), j, d) \quad \text{if } u \neq r_{x,y} \text{ or } v \neq t;$$

$$\Delta(tokenF(r_{x,y}, i, p_x, t, j, d)): \exists h \exists d' (tokenF(r_{x,y}, i, p_x, t_1, h, d') \wedge Futr(tokenF(t_1, h, q, t_2, j, d - d'), d')).$$

In words, the token produced by the transition $r_{x,y}$ and consumed by t after d time units corresponds in net I to two tokens, one produced by $r_{x,y}$ and consumed by

t_1 after d' and the other produced by the same firing of t_1 and consumed by t_2 after d time units.

We prove for this rule that: $\vdash_{\mathscr{I}} \Delta(LB_S(t))$, where

$$LB_S(t): \quad fireth(t,i) \rightarrow \bigwedge_{j=1}^{p} \exists d \exists x \left(d \geqslant m_t \wedge \bigvee_{k=1}^{n_j} tokenP(r_{j,k}, x, p_j, t, i, d) \right)$$

The translation of the *tokenP* predicate, as deriving from definition (DEF), and from the proposed translation of *tokenF* is

$$\begin{aligned} \Delta(tokenP(r_{x,y}, i, p_x, t, j, d)) &= \Delta(Past(tokenF(r_{x,y}, i, p_x, t, j, d), d)) \\ &= \exists h \exists e (tokenP(t_1, h, q, t_2, j, e) \\ &\quad \wedge Past(tokenP(r_{x,y}, i, p_x, t_1, h, d-e), e)). \end{aligned}$$

Then $\Delta(LB_S(t))$ is

$$\begin{aligned} fireth(t_2, i) \rightarrow \bigwedge_{j=1}^{p} \exists d \exists x \Bigg(& d \geqslant m_t \wedge \bigvee_{k=1}^{n_j} \exists h \exists e (tokenP(t_1, h, q, t_2, i, e) \\ & \wedge Past(tokenP(r_{j,k}, x, p_j, t_1, h, d-e), e)) \Bigg). \end{aligned}$$

To prove this we apply the deduction theorem, i.e., we derive the conclusion of the implication assuming its premise. First, we assume a firing of t_2.

1. $fireth(t_2, i)$

Applying $LB(t_2)$ (for net I) and PI (see Note 4) we derive

2. $\exists h \exists d' (d' \geqslant m_{t_2} \wedge tokenP(t_1, h, q, t_2, i, d') \wedge Past(fireth(t_1, h), d'))$.

Thanks to $LB(t_1)$ and some tautologies and first-order theorems we get

3. $\bigwedge_{j=1}^{p} \exists h \exists e \exists f \exists x (d' \geqslant m_{t_2} \wedge tokenP(t_1, h, q, t_2, i, e) \wedge f \geqslant m_{t_1}$

$\wedge \bigvee_{k=1}^{n_j} Past(tokenP(r_{j,k}, x, p_j, t_1, h, f), e))$.

From the refinement rule (Table 2((d)) we have $m_t = m_{t_1} + m_{t_2}$; if we take $d = f + e$ the thesis follows.

4. $\bigwedge_{j=1}^{p} \exists d \exists x (d \geqslant m_t \wedge \exists e \exists h (tokenP(t_1, h, q, t_2, i, e)$

$\wedge Past(tokenP(r_{j,k}, x, p_j, t_1, h, d-e), e))$.

5.1.3. Transition splitting rule

The translation of the *tokenF* predicate in this case is not immediately defined as in the preceding rules. All trivial translations (which for brevity we do not discuss here: the interest reader is referred to [13]) are not adequate to prove the correctness of the

rule, either because the sentences obtained by translation of the axioms of net S are either too restrictive on the behavior of net I (and hence false) or because they happen to be totally unrelated with it. It turns out that to permit proof of correctness of the transition splitting rule the Δ function should be defined in such a way that it sets a one-to-one correspondence between the firings of transition t on the one side and those of transitions t_1 or t_2 on the other side.

One definition of function Δ on the *tokenF* predicates that satisfies this constraint is the following.

$$\Delta(tokenF(u,i,p,v,j,d)) = tokenF(u,i,p,v,j,d) \text{ if } u \neq t \text{ and } v \neq t;$$

$$\Delta(tokenF(u,i,p,t,j,d))$$

$$= \exists n_1 \left(Futr(nFire(t,n_1),d) \wedge \left(\begin{array}{c} j \leqslant n_1 \wedge tokenF(u,i,p,t_1,j,d) \\ \vee \\ j > n_1 \wedge tokenF(u,i,p,t_2,j-n_1,d) \end{array} \right) \right),$$

$$\Delta(tokenF(t,i,p,v,j,d))$$

$$= \exists n_1 \left(nFire(t,n_1) \wedge \left(\begin{array}{c} i \leqslant n_1 \wedge tokenF(t_1,i,p,v,j,d) \\ \vee \\ i > n_1 \wedge tokenF(t_2,i-n_1,p,v,j,d) \end{array} \right) \right).$$

This translation states, arbitrarily but without loss of generality, that the firings of transition t_1 correspond to the ones of t having a lowest index, while those of t_2 correspond to those with highest index.

For this rule we report the proof of $\vdash_{\mathscr{I}} \Delta(OU_S(r_{x,y}))$, where

$$OU_S(r_{x,y}): \quad tokenF(r_{x,y},i,p_i,t,j,d) \wedge tokenF(r_{x,y},i,p_x,t,m,e) \rightarrow j = m \wedge d = e$$

and

$$\Delta(OU_S(r_{x,y})) =$$

$$\left(\begin{array}{l} \exists n_1 \left(Futr(nFire(t_1,n_1),d) \wedge \left(\begin{array}{c} j \leqslant n_1 \wedge tokenF(r_{x,y},i,p_x,t_1,j,d) \\ \vee \\ j > n_1 \wedge tokenF(r_{x,y},i,p_x,t_2,j-n_1,d) \end{array} \right) \right) \\ \wedge \\ \exists m_1 \left(Futr(nFire(t_1,m_1),e) \wedge \left(\begin{array}{c} m \leqslant m_1 \wedge tokenF(r_{x,y},i,p_x,t_1,m,e) \\ \vee \\ m > m_1 \wedge tokenF(r_{x,y},i,p_x,t_2,m-m_1,e) \end{array} \right) \right) \end{array} \right)$$

$$\rightarrow \begin{pmatrix} j = m \\ \wedge \\ d = e \end{pmatrix}$$

Again we use the deduction theorem to derive the conclusion assuming the premise. In the premise, let n_1 be N_1 and m_1 be M_1 (i.e., we apply the existential instantiation theorem); then, from the $\wedge$-elimination rule ($\alpha \wedge \beta \vdash \alpha$) we get

1. $(tokenF(r_{x,y},, p_x, t_1, j, d) \vee tokenF(r_{x,y}, i, p_x, t_2, j - N_1, d))$

   $\wedge (tokenF(r_{x,y}, i, p_x, t_1, m, e) \vee tokenF(r_{x,y}, i, p_x, t_2, m - M_1, e))$

   Applying the distributive property of conjunction we get:

2. $tokenF(r_{x,y}, i, p_x, t_1, j, d) \wedge tokenF(r_{x,y}, i, p_x, t_1, m, e)$

   $\vee\ tokenF(r_{x,y}, i, p_x, t_1, j, d) \wedge tokenF(r_{x,y}, i, p_x, t_2, m - M_1, e)$

   $\vee\ tokenF(r_{x,y}, i, p_x, t_2, j - N_1, d) \wedge tokenF(r_{x,y}, i, p_x, t_1, m, e)$

   $\vee\ tokenF(r_{x,y}, i, p_x, t_2, j - N_1, d) \wedge tokenF(r_{x,y}, i, p_x, t_2, m - M_1, e)$

The second and the third disjoints are false, because a token produced by $r_{x,y}$ in p_x cannot be consumed by both t_1 and t_2. Thanks to $OU(r_{x,y})$, the first disjoint implies $d = e \wedge j = m$ and the last disjoint implies $d = e \wedge j - N_1 = m - M_1$. From axiom $UFN(t_1)$, the number of firings of t_1 is unique, i.e. $M_1 = N_1$. Then we can derive

3. $d = e \wedge j = m$.

5.1.4. Iteration rule

The iteration rule can be proven correct by defining the Δ function on the tokenF predicate as follows:

$$\Delta(tokenF(u, i, p, v, j, d)) = tokenF(\lambda^{-1}(u), i, p, \lambda^{-1}(v), j, d) \quad \text{if } u \neq r_{x,y} \text{ or } v \neq t;$$

$$\Delta(tokenF(r_{x,y}, i, p_x, t, j, d))$$

$$= \exists h' \exists d' \exists h'' \exists d'' \begin{pmatrix} tokenF(r_{x,y}, i, p_x, t_1, h', d') \wedge \\ Futr(tokenF(t_1, h', q_1, t_3, h'', d''), d') \wedge \\ Futr(tokenF(t_3, h'', q_5, t_5, j, d - d' - d''), d' + d'') \end{pmatrix}$$

In words, a token produced in net S by the transition $r_{x,y}$ and consumed by t after d time units corresponds, in net I, to three tokens, the first one produced by $r_{x,y}$ and consumed by t_1 after d', the second one produced by the same firing of t_1 and consumed by t_3 after d'' time units and the third one produced by the same firing of t_3 and consumed after d time units from the transition t_5 firing.

For this rule the most interesting proof is that regarding the upper bound property for the refined transition, i.e., the proof that $\vdash_{\mathscr{I}} \Delta(UB_S(t))$. This is by no means a trivial

exercise, due both to the topological complexity of the refined net, and to the fact that, since no boundedness hypothesis is assumed for the net, tokens produced by distinct firings of the same transition may interfere.

Here we give only a brief trace of the proof, which follows immediately from the following four main lemmas.

Lemma 1 asserts that if t_1 fires, either there is a token produced by t_4 in q_2 at some time in the next time interval $Mt_4 + ex$ time units long, or t_5 fires and consumes the token in q_2 within the end of that interval.

Lemma 2 asserts that if transition t_4 (or t_1) fires after a firing of t_3, then either transition t_5 fires immediately and consumes the token in q_2, or else the token produced by t_3 was already consumed by t_5 before the firing of t_4 (or t_1).

Lemma 3 asserts that if t_3 fired $ex + Mt_4$ time units ago and t_1 fired sometimes in the past then at least one of the tokens produced by these firings was consumed by t_5 before now. This can be justified as follows: thanks to Lemmas 1 and 2 at some instant between the assumed firing of t_3 and now there is a token in p_2 produced or by t_1 or by t_4, so t_5 fires and consumes at least one of the two tokens.

Lemma 4 asserts that any token produced by a firing of t_3 is consumed by a firing of t_5 within $ex + M_{t_4}$ time units. The proof of this lemma is based on Lemma 3 and involves considering the so-called *non Zeno property*, asserting that the number of firings in a limited time interval is finite; in fact, if this lemma did not hold we would have a infinite number of firings of t_1 from the start of the net execution to the current time.

5.1.5. Place sequencing rule

This rule, shown in Table 3(b) is very similar to the transition sequencing rule of Table 2(d). In fact, it could be proven correct using Corollary 8 as in the above discussed cases, defining Δ as follows:

$$\Delta(tokenF(u, i, p, v, j, d)) = tokenF(\lambda^{-1}(u), i, p, \lambda^{-1}(v), j, d)$$

$$\text{if} \quad u \neq r_x \text{ and } v \neq t_y;$$

$$\Delta(tokenF(r_x, i, p, t_y, j, d)) = \exists h\, \exists d'(tokenF(r_x, i, p, s, h, d') \wedge Futr(tokenF(s, h, q, t_y, \mathrm{j}, d - d'), d')).$$

We do not develop further the proof of correctness for the place sequencing rule, however, because place sequencing rule can be seen as a particular case of process splitting rule, which will be treated in Section 5.2.

5.2. An extension to the proof method

The method for proving correctness of refinement rules, based on Corollary 8 and applied to several cases in the preceding Section 5.1, does not apply to the place splitting and process splitting rules.

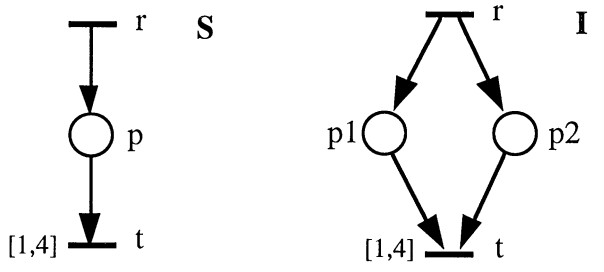


Fig. 5. A simplest application of the place splitting refinement rule.

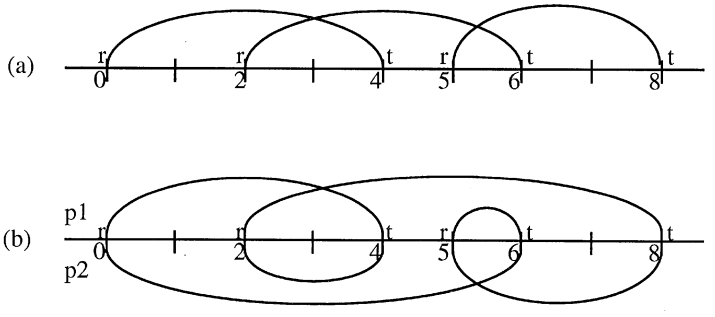


Fig. 6. Token production and consumption in the two nets S (a) and I (b) of Fig. 5.

We illustrate this fact by referring, without loss of generality, to the simplest nets of Fig. 5, where we assume the time bounds of transition t to be $m_t = 1$ and $M_t = 4$. We consider, as shown in Fig. 6(a), an execution of net S where transition r fires at times 0, 2, and 5, while transition t fires at times 4, 6, and 8. The lines connecting transition firings above the time axis show that the tokens produced by r's firing at times 0, 2, and 5 are consumed respectively at times 4, 6, and 8 by a firing of transition t. Fig. 6(b) shows an execution of net I where transitions r and t fire exactly at the same times as the homonymous transitions in S; the lines above (resp., below) the time axis describe the production and consumption of tokens passing through place p_1 (resp., p_2): for instance, the token introduced into p_1 by the firing of r at time 2 is consumed by the firing of t at time 8, while the token introduced by the same firing of r at time 2 into place p_2 is consumed by the firing of t at time 4.

First, we illustrate how the three simple (tentative) proof functions $\Delta 1$, $\Delta 2$, and $\Delta 3$ reported in Table 4 do not satisfy the requirement (ii) of Corollary 8 (i.e., not all axioms of $Ax(S)$, translated through them, are theorems of net I). Let us consider the predicate $tokenF(r, i, p, t, j, d)$ that relates firings of transitions r and t, with the token produced by r's firing consumed d time units later by a firing of t.

Proof function $\Delta 1$, applied to axiom

$$UB_S(t)\text{: } fireth(r, i) \rightarrow \exists d(d \leqslant M_t \wedge \exists j\, tokenF(r, i, p, t, j, d))$$

Table 4
Proof functions for the place splitting rule

Δ	F	$\Delta(F)$
$\Delta 1$	$tokenF(r, i, p, t, j, d)$	$tokenF(r, i, p_1, t, j, d) \wedge tokenF(r, i, p_2, t, j, d)$
$\Delta 2$	$tokenF(r, i, p, t, j, d)$	$tokenF(r, i, p_1, t, j, d)$
$\Delta 3$	$tokenF(r, i, p, t, j, d)$	$tokenF(r, i, p_1, t, j, d) \vee tokenF(r, i, p_2, t, j, d)$

returns the formula

$$\Delta 1(UB_S(t)) = fireth(r, i) \rightarrow \exists d(d \leqslant M_t \wedge \exists j(tokenF(r, i, p_1, t, j, d) \wedge tokenF(r, i, p_2, t, j, d))).$$

which asserts that in net I two tokens introduced into places p_1 and p_2 by one given firing of transition r would always be consumed by the same firing of transition t. This is however false, as it is shown by all firings of transition r in Fig. 6(b). Hence, $\Delta 1$ is not adequate for proving correctness of the place splitting refinement rule because, in a sense, it imposes too strong a constraint on I's behavior.

The proof function $\Delta 3$ suffers symmetrical problems, with axiom

$$OU_S(r): \quad tokenF(r, i, p, t, j, d) \wedge tokenF(r, i, p, t, k, e) \rightarrow j = k \wedge d = e$$

that, translated through $\Delta 3$, becomes

$$\Delta 3(OU_S(r)) = \left(\begin{array}{c} (tokenF(r, i, p_1, t, j, d) \vee tokenF(r, i, p_2, t, j, d)) \\ \wedge \\ (tokenF(r, i, p_1, t, k, e) \vee tokenF(r, i, p_2, t, k, e)) \end{array} \right) \rightarrow j = k \wedge d = e$$

which implies

$$tokenF(r, i, p_1, t, j, d) \wedge tokenF(r, i, p_2, t, k, e) \rightarrow j = k \wedge d = e$$

again asserting that in net I two tokens introduced into places p_1 and p_2 by one given firing of transition r are necessarily consumed by the same firing of transition t.

As a final example, proof function $\Delta 2$ applied to $UB_S(t)$ produces the following formula

$$\Delta 2(UB_S(t)) = fireth(r, i) \rightarrow \exists d(d \leqslant M_t \wedge \exists j\, tokenF(r, i, p_1, t, j, d))$$

asserting that any token introduced into p_1 by a firing of transition r will always be consumed by a firing of transition t within t's upper bound M_t, which is clearly false, as shown by r's firing at time 2 in Fig. 6(b).

We can understand why any attempt to prove correctness of the place splitting rule by using a proof function is bound to fail by considering once more the execution of net S depicted in Fig. 6(a) and confronting it with that of net I in Fig. 6(b). The use of a proof function Δ for proving correctness of a refinement rule is based on the idea

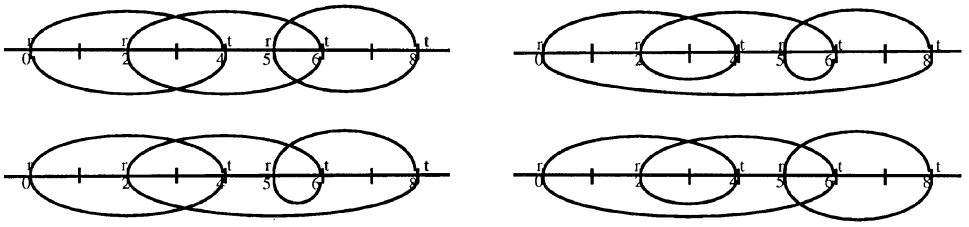


Fig. 7. Some more executions of net I.

that the translation of the *tokenF* predicate through Δ describes how the mechanism of token production and consumption by transitions of S is simulated in net I. Several applications of this idea were discussed in Section 5.1. Applying this approach to the place splitting refinement rule amounts to trying to translate the production of a token by r and its consumption by t into the production and consumption of some tokens in net I. The example of net executions in Fig. 6 shows, however, that this in not possible. In fact, in Fig. 6(a) neither the tokens flowing through place p_1 nor those flowing through p_2 in net I behave like the tokens flowing through the unique place p in net S. (Notice that there are *other* executions of net I, some of which are shown in Fig. 7, where either the tokens flowing through p_1 or through p_2 of net I or both follow the same pattern as those in place p of net S, but these are not the only possible executions.)

In conclusion, no proof function satisfying the hypotheses of Corollary 8 can be defined for the place splitting rule. A similar reasoning applies to the process splitting refinement rule, where the amount of nondeterminism introduced by the refinement step is even greater.

To overcome these difficulties we propose a proof method that extends the one defined by Corollary 8 and adopted so far.

Theorem 9. *Let S, I, $\mathscr{S}$, $\mathscr{I}$, and Λ be defined as in Corollary* 8. *Assume that $\Delta 1$ and $\Delta 2$ are two proof functions from $\mathscr{S}$ to $\mathscr{I}$ such that*

$$(*) \quad \vdash_{\mathscr{I}} \Delta 1(Ax(S)) \vee \Delta 2(Ax(S)).$$

Then net I implements net S.

Proof. Let φ be an observable property of net S, i.e., $\vdash_{\mathscr{S}} \varphi$. By Metatheorem 7, we have

$$(\bullet) \quad \Delta 1(Ax(S)) \vdash_{\mathscr{I}} \Delta 1(\varphi) \quad \text{and} \quad \Delta 2(Ax(S)) \vdash_{\mathscr{I}} \Delta 2(\varphi).$$

Let $\Delta 1(Ax(S)) = \{\phi_1, \ldots, \phi_n\}$ and $\Delta 2(Ax(S)) = \{\psi_1, \ldots, \psi_n\}$; then $(\bullet)$ can be rewritten as

$$\{\phi_1, \ldots, \phi_n\} \vdash_{\mathscr{I}} \Delta 1(\varphi) \quad \text{and} \quad \{\psi_1, \ldots, \psi_n\} \vdash_{\mathscr{I}} \Delta 2(\varphi).$$

Applying repeatedly the deduction theorem we get

$$\vdash_{\mathscr{I}} \phi_1 \to \cdots \to \phi_n \to \Delta 1(\varphi) \quad \text{and} \quad \vdash_{\mathscr{I}} \psi_1 \to \cdots \to \psi_n \to \Delta 2(\varphi)$$

and, by the tautology $(\alpha \to \beta \to \gamma) \to (\alpha \wedge \beta \to \gamma)$,

$$\vdash_{\mathscr{I}} \phi_1 \wedge \cdots \wedge \phi_n \to \Delta 1(\varphi) \quad \text{and} \quad \vdash_{\mathscr{I}} \psi_1 \wedge \cdots \wedge \psi_n \to \Delta 2(\varphi).$$

Since $\Delta 1$ and $\Delta 2$ are extensions of Λ, it follows that

$$\vdash_{\mathscr{I}} \phi_1 \wedge \cdots \wedge \phi_n \to \Lambda(\varphi) \quad \text{and} \quad \vdash_{\mathscr{I}} \psi_1 \wedge \cdots \wedge \psi_n \to \Lambda(\varphi)$$

from which, thanks to hypothesis $(*)$ and Case Analysis we finally get the thesis, $\vdash_{\mathscr{I}} \Lambda(\varphi)$. □

Remark. Notice that using two proof functions $\Delta 1$ and $\Delta 2$ when applying Theorem 9 is *not* equivalent to using a single proof function Δ defined as $\Delta(\varphi) = \Delta 1(\varphi) \vee \Delta 2(\varphi)$. In fact, it can be easily verified that function Δ as just defined is not even a proof function, because $\Delta(\beta \wedge \gamma) \neq \Delta(\beta) \wedge \Delta(\gamma)$.

Theorem 9 can be immediately generalized to the case where, instead of two proof functions $\Delta 1$ and $\Delta 2$, a generic positive number n of proof functions $\Delta 1, \ldots, \Delta n$ are used.

Corollary 10 (To Theorem 9). *In the same hypotheses as Theorem* 9, *assume that n proof functions* $\Delta 1, \ldots, \Delta n$ *from* $\mathscr{S}$ *to* $\mathscr{I}$ *are given, such that*

$$\vdash_{\mathscr{I}} \Delta 1(Ax(S)) \vee \Delta 2(Ax(S)) \vee \cdots \vee \Delta n(Ax(S)).$$

Then net I implements net S.

The proof of Corollary 10 is a straightforward generalization of the proof of Theorem 9.

Theorem 9 and Corollary 10 can be used to prove the correctness of the place splitting and process splitting refinement rules.

As in Section 5.1, we consider that $Ax(S) = UFN_S \cup NNF_S \cup FI_S \cup LB_S \cup UB_S \cup IU_S \cup OU_S \cup IM_S$, and that the only nontrivial proofs are those for the topological axioms of the transitions involved in the refinement. Furthermore, the places involved in the application of the refinement rule are empty in the initial marking. Therefore, the correctness proof amounts to

$$\vdash_{\mathscr{I}} \bigvee_{i=1}^{s} (\Delta_i(LB(t_x)) \wedge \Delta_i(UB(t_x)) \wedge \Delta_i(OU(r_y)) \wedge \Delta_i(IU(t_x))$$

In both the place and process splitting cases the proof is divided in to two steps. As a first step it is immediate to derive:

$$\vdash_{\mathscr{I}} \bigwedge_{k=1}^{s} (\Delta_k(LB(t_x)) \wedge \Delta_k(OU(r_y)) \wedge \Delta_k(IU(t_x))).$$

As a second step, it therefore remains to prove that $\vdash_{\mathscr{I}} \bigvee_{i=1}^{s} \Delta_i(UB(t_x))$.

Again, for the sake of brevity we only present here the definition of the proof functions, and report the details of the proofs in [8].

5.2.1. Place splitting rule

The proof of correctness uses two proof functions $\Delta 1$ and $\Delta 2$. For transitions not in the preset nor in the postset of the refined place, i.e., for $u \neq r_h$, $\forall h \in [1..n]$, or $v \neq t_k$, $\forall k \in [1..m]$, they are defined as follows.

$$\Delta 1(tokenF(u, i, q, v, j, d)) = \Delta 2(tokenF(u, i, q, v, j, d))$$
$$= tokenF(\lambda^{-1}(u), i, q, \lambda^{-1}(v), j, d).$$

For transitions in the preset or postset of the refined place, $\Delta 1$ and $\Delta 2$ are reported below.

$$\Delta 1(tokenF(r_h, i, p, t_k, j, d)) = tokenF(r_h, i, p_1, t_k, j, d) \quad \forall h \in [1..n], \forall k \in [1..m];$$

$$\Delta 2(tokenF(r_h, i, p, t_k, j, d)) = tokenF(r_h, i, p_2, t_k, j, d) \quad \forall h \in [1..n], \forall k \in [1..m].$$

$\Delta 1$ and $\Delta 2$ are therefore equal except for the way they translate the production and consumption of tokens flowing through the refined place p: $\Delta 1$ asserts that in net I the token flows through place p_1, while $\Delta 2$ through place p_2.

5.2.2 Process splitting rule

For the process splitting rule we can apply Corollary 10 with as many Δi's as there are branches in the refined net fragment. Assuming, as in Table 3(d), that this number is s, in the following we define Δ_h, for each $h \in [1..s]$, as follows. As in the preceding paragraph we treat differently the case of transitions in the preset and postset of the refined place:

$$\Delta_h(tokenF((u, i, q, v, j, d)) = tokenF(\lambda^{-1}(u), i, q, \lambda^{-1}(v), j, d)$$
$$\text{for} \quad u \neq r_k, \forall k \in [1..n], \quad \text{or} \quad v \neq t_l, \forall l \in [1..m]$$

$$\Delta_h(tokenF(r, i, p, t_1, j, d)) = \exists e \exists x (tokenF(r_k, i, p_h, v_h, x, e)$$
$$\wedge Futr(tokenF(v_h, x, q_h, t_1, j, d - e), e)) \quad \forall k \in [1..n], \forall l \in [1..m].$$

Again the difference among the various Δ_h's is limited to the way they translate the production and consumption of tokens flowing through the refined place p: each different Δ_h, for $h \in [1..s]$, asserts that in net I the token traverses the branch with index h, flowing through places p_h and q_h.

6. Incremental analysis

Implementation through refinements is not only important as a means of structured development, but it also enhances the analyzability of the specifications by promoting their incremental analysis, thanks to the possibility to inherit the properties already proven at more abstract levels. The inherited properties can be used as lemmas for proving new properties at the implementation level that cannot be proven inside the specification, e.g., because they refer to transitions that do not appear at the specification level.

We illustrate these concepts through the system specified and refined in Figs. 3 and 4. We outline an incremental proof of the following property for net N_6 of Fig. 3: any firing of transition t_1 will always be followed within 10 time units by a firing of t_{511}. This property is expressed by the following formula:

$$Alw(\exists i\ fireth(t_1, i) \rightarrow WithinF(\exists j\ fireth(t_{511}, j), 10)).$$

To simplify the notation and improve the readability of formulas we adopt the convention of writing, for any transition t, the formula $fire(t)$ with the meaning $\exists i$ $fireth(t, i)$. Property P can thus be restated as

$$(\ddagger) \qquad Alw(fire(t_1) \rightarrow WithinF(fire(t_{511}), 10))$$

Based on the Temporal Generalization theorem we limit ourselves to deriving the formula argument of the outermost Alw operator, i.e. "$fire(t_1) \rightarrow WithinF(fire(t_{511}), 10)$".

1. The first lemma in the proof refers to net N_1 and states that after the firing of t_1, t_3 will eventually fire within 5 time units:

P1: $fire(t_1) \rightarrow WithinF(fire(t_3), 5)$

The proof of P_1 is based on the axioms $LB(t_1)$, $UB(t_1)$, $LB(t_3)$, and $UB(t_3)$, and uses the case analysis rule. From $fire(t_1)$ it can be derived, by $LB(t_1)$ and $UB(t_1)$, $Past(WithinP(fire(itp_1) \vee fire(t_2), 2), 4)$. The less favorable case is that corresponding to the most recent time for the firing of t_2 or itp_2: from $Past(fire(itp_1) \vee fire(t_2), 4)$ we derive $Past(fire(itp_2) \vee fire(t_2), 4)$ by case analysis: this because $Past(fire(itp_1), 4)$

[8] Strictly speaking, the axioms describing the initial marking are not temporally closed, so the formulas derived by means of their application cannot be generalized through the Temporal Generalization metatheorem. In this case, however, instead of using the $IM(p_1)$ and $IM(p_2)$ axioms, the derivation can employ the following formula:

$$Alw(AlwP(nFire(itp_1, 0) \wedge nFire(itp_2, 0)) \wedge \exists t(Futr(nFire(itp_1, 1) \wedge nFire(itp_2, 1) \wedge AlwF(nFire(itp_1, 0) \wedge nFire(itp_1, 0)), t))$$
$$\vee IM(p_1) \wedge IM(p_2) \vee$$
$$AlwF(nFire(itp_1, 0) \wedge nFire(itp_2, 0)) \wedge \exists t(Past(nFire(itp_1, 1) \wedge nFire(itp_2, 1) \wedge AlwP(nFire(itp1, 0) \wedge nFire(itp1, 0)), t))).$$

This formula essentially asserts that transitions itp_1 and itp_2 fire just once and at the same time, which time may be in the past, now or in the future. Such a formula is temporally closed and immediately derivable from $IM(p_1)$ and $IM(p_2)$.

implies $Past(fire(itp_2), 4)$ since transitions itp_1 and itp_2 fire just once, at the same time, as described by the axioms for the initial marking $IM(p_1)$ and $IM(p_2)$.[8] From $Past(fire(itp_2) \vee fire(t_2), 4)$, using $LB(t_3)$ and $UB(t_3)$, we derive $Past(Futr(WithinF(fire(t_3), 3), 6), 4)$, which implies $WithinF(fire(t_3), 5)$.

2. Then in net N_2 we are able to prove the property P_2: $fire(t_1) \rightarrow WithinF(fire(t_{21}), 6)$ using Λ_1(P1) as a lemma and $UB(t_{21})$. This property establishes an upper bound to the firing of transition t_{21} after that of t_1.

3. In the net N_3 we can prove property P3 that establishes an upper bound for the time elapsing from the firing of t_{21} until the firing of t_{51}, using axiom $UB(t_{51})$.

P3: $fire(t_{21}) \rightarrow WithinF(fire(t_{51}), 6)$.

4. Property P3 is inherited in N_6 as

$$\Lambda_5(\Lambda_4(\Lambda_3(\text{P3}))) = fire(t_{21}) \rightarrow WithinF(fire(t_{511}) \vee fire(t_{512}), 6).$$

5. In N_6 we can now show that the time bounds on transitions t_{511} and t_{512} resolve the non-determinism introduced by the application of the transition splitting rule.

P4: $fire(t_{21}) \rightarrow WithinF(fire(t_{511}), 6)$.

P4 is easily proved using as lemmas $\Lambda_5(\Lambda_4(\Lambda_3(\text{P3})))$ and property P5: $Alw(\neg fire(t_{512}))$, whose proof is reported, for a similar net, in [12].

6. In net N_6 we now derive a stronger constraint on the time bound among the firing of transition t_{511} after that of t_{21}.

P6: $fire(t_{21}) \rightarrow WithinF(fire(t_{511}), 4)$

P6 is a straightforward consequence of P5 and of the net axiom $UB(t_{511})$.

Finally P6, together with $\Lambda_5(\Lambda_4(\Lambda_3(\Lambda_2(\text{P2}))))$, allows us to prove the desired property P.

To fully appreciate the simplification introduced by our incremental approach to the proof of net properties the reader should consider, for instance, that the derivation of the first lemma P1, which used the axiom $UB(t_3)$ for net N_1, would be quite intricate if conducted at the level of the net N_6; in that case, each application of the net axiom $UB(t_3)$ should be substituted by a proof similar to the one for $\Delta(UB_S(t))$ (where Δ is the proof function for the iteration refinement rule) included in [8].

7. Conclusions

The principal motivation for our research lies in the belief that formal methods and the related specification and verification techniques provide an adequate theoretical basis and a useful methodological support to the development of time critical systems. The use of formal specification and verification techniques can improve the reliability of time critical systems by permitting the production of unambiguous specification and by supporting the use of powerful tools to detect faults early in the development process.

In our view, one of the reasons why formal methods are still seldom applied in the industrial practice is that for most formalisms having high expressive power the verification procedures either cannot be performed mechanically or their computational cost is too high to permit their use in realistic projects. These problems can be addressed by adopting a development methodology based on successive refinements that reduces the overall design and verification effort by reusing, at each phase of the development, the results gathered in the preceding steps.

In the present paper we discussed these issues referring to an operational formalism like timed Petri nets used as abstract system models, and to the TRIO temporal logic used as a means to describe desired timing properties. We formally defined the notions of implementation and provided a general method to prove that a set of refinement rules are correct with respect to this notion of implementation. We also showed how the development of a system through a series of correct refinement steps greatly facilitates its verification by allowing the designer to prove with a reduced effort intermediate lemmas on the early versions of the system. The described refinement rules are now supported by Cabernet [23], a tool for the incremental specification and design of time critical systems based on timed Petri nets developed at Politecnico di Milano.

We claim that the presented development method, with its related formal definitions and proofs, can be adapted, with suitable modifications, to other formalisms that combine a descriptive language for specifying timing requirements with an operational notation to model system structure, such as, for instance, timed transition systems [17] or TTM/RTTL [22].

As noted in the introduction, our logic-based characterization of the implementation relation differs from other works on refinement, where such definitions are based on behaviors and execution traces. It is therefore interesting and useful to provide an alternative, more "operational" view of our notion of implementation. The semantics of timed Petri nets is often defined in terms of *observable time behaviors*: an observable time behavior (OTB for short) lists all the transition firings in a given execution of the net, each firing being associated with the time of its occurrence. According to this view, the firings are the only externally observable elements of the net, whose semantics is defined as the set of all possible OTBs. Then a natural definition of the implementation relation [9] states that a net I implements net S iff the set of OTB of I is a *subset* of the set of OTB of S. It can be immediately noticed that this notion of refinement is equivalent to the one presented in this paper, by considering that every OTB essentially identifies (i.e., is in one-to-one correspondence with) a model of the TRIO formulas that describe the net behavior. Therefore, requiring that the set of OTB (that is, of the models of the TRIO formulas) of net I be a subset of those of S (after a possible renaming of transitions) is equivalent to requiring that all formulas satisfied in the models of S be satisfied (again, after a possible renaming of the transitions) also in the models of I, as we do in the present work.

Yet another way of defining implementation among timed Petri nets could be based on the notion (very common in the literature on TPNs) of *simulation*, where a relation

among the states of two nets is introduced, such that if the two nets are in related states and one net executes a transition that takes it to another state, then the other, simulating net can execute the same transition (or a similar one) and enter a related state. Characterization of implementation among TPNs in terms of simulation is an interesting open research problem. Any work in this direction should take into account the fact that the notion of state for TPNs is more elaborate than for untimed Petri nets: as shown in [5], the state space of any nontrivial timed Petri net has an uncountable infinite set of states, which can be grouped (under some easily-satisfied conditions) into a finite or denumerable set of state classes. It is to be expected that the definition of implementation in terms of simulation would require the specification net S and the implementation net I to have state graphs with some kind of structural relationship.

References

[1] M. Abadi, L. Lamport, The existenece of refinement mappings, Theoret. Comput. Sci. 82 (1991) 253–284.
[2] J.I. Aizikowitz, Designing distributed services using refinement mappings, Ph.D. Thesis and Tech. Report 89-1040, Cornell University, Ithaca, New York, 1990.
[3] R. Alur, T.A. Henzinger, Real time logics: complexity and expressiveness, Tech. Report no. STANCS901307, IEEE LICS'90 (1990, pp. 390–401).
[4] K. Apt, Ten years of Hoare's lLogic: a survey – Part I, ACM-Trans. Programming Languages Systems 3 (4) (1981) 431–483.
[5] B. Barthomieu, M. Diaz, Modeling and verification of time dependent systems using time petri nets, IEEE TSE-Trans. on Software Eng. SE-17, March (1991) 259–273.
[6] W. Damm, G. Dohmen, V. Gerstner, B. Josko, Modular verification of Petri nets, the temporal logic approach, in: Proc. Stepwise Refinement of Distributed Systems. Models, Formalisms, Correctnesss, Lecture Notes in Computer Science, Vol. 430, Springer, Berlin, 1990, pp. 181–207.
[7] H.B. Enderton, A Mathematical Introduction to Logic, Academic Press, New York: 1972.
[8] M. Felder, A. Gargantini, A. Morzenti, A theory of implementation and refinement in timed petri nets, Tech. Report 95-044, Politecnico di Milano, Dipartimento di Elettronica e Informazione, Sept. 1995 (also available via anonymous ftp on the site ftp-se.elet.polimi.it as file /dist/Papers/FormalMethods/DualLanguage/TechRep95-044.ps.z).
[9] M. Felder, C. Ghezzi, M. Pezzè, Analyzing refinements of state based specifications: the case of TB nets, in: Proc. ISSTA'93, Cambridge, 1993, pp. 28–39.
[10] M. Felder, A. Morzenti, Validating real-time systems by executing logic specifications in TRIO, in: Proc. 14th Internat. Conf. on Software Engineering, ACM/IEEE, 1992, pp. 199–211.
[11] M. Felder, D. Mandrioli, A. Morzenti, Proving properties of real-time systems through logical specifications and Petri nets models, Tech. Rep., TR 91-072, Diparimento di Elettronica e Informazione, Politecnico di Milano, December 1991.
[12] M. Felder, D. Mandrioli, A. Morzenti, Proving properties of real-time systems through logical specifications and Petri net models, IEEE TSE-Trans. Software Eng. 20 (1994) 127–141.
[13] A. Gargantini, A temporal logic-based approach to the implementation and refinement of timed Petri nets, Thesis, Politecnico di Milano, Dipartimento di Elettronica, 1995 (in Italian).
[14] R. Glabbeek, U. Goltz, Refinement of actions in causality based models, in: Proceedings of Stepwise Refinement of Distributed Systems. Models, Formalisms, Correctnesss, Lecture Note in Computer Science, Vol. 430, Springer, Berlin, 1990, pp. 266–300.
[15] C. Ghezzi, M. Jazayeri, D. Mandrioli, Fundamentals of Software Engineering, Prentice-Hall, Englewood Cliffs, N.J., 1991.

[16] C. Ghezzi, D. Mandrioli, A. Morzenti, TRIO, a logic language for executable specifications of real-time systems, J. Systems Software 12 (1990) 107–123.
[17] T. Henzinger, Z. Manna, A. Pnueli, Temporal proof methodologies for real-time systems, in: Proc. 18th Ann. Symp. on Principles of Programming Languages, ACM Peress, New York 1991, pp. 353–366.
[18] N. A. Lynch, H. Attiya, Using mapping to prove timing properties Tech. Report MIT/LCS/TM-412,b Laboratory for Computer Science, MIT, 1989. Appeared in Proc. PODC'90.
[19] E. Mendelson, Introduction to Mathematical Logic, Van Nostrand Reinold Company, New York, 1963.
[20] P.M. Merlin, D.J. Farber, Recoverability of communication protocols – implications of a theoretical study, IEEE Trans. Commun. 24 (1976) 1036–1043.
[21] K. Müller, Constructable Petri nets, in: Proc. EIK 21, 1985, pp. 17–199.
[22] J. Ostroff, Temporal Logic For Real-Time Systems, Advanced Software Development Series, 1, Research Studies Press LTD., Taunton, Somerset, England: 1989.
[23] G. Pezzè, Cabernet: a customizable environment for the secification and analysis of real-time systems, in: Proc. DECUS Europe Symp., 1992.
[24] A. Pnueli, Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends, Lecture Notes in Computer Science, Vol. 224, Springer, Berlin, 1986.
[25] W. Reisig, Petri Nets: An Introduction, EATCS Monographs on Theoretical Computer Science, Springer, Berlin, 1985.
[26] I. Suzuki, T. Murata, A method of stepwise refinement and abstraction of Petri nets, J. Comput. System Sci. (18) (1979) 35–46.
[27] W. Vogler, Behaviour preserving refinements in Petri nets, in: Proc. 12th Internat Worksop on Graph Theoretic Concepts in Computer Science, München, 1986, Lecture Notes in Computer Science, Vol. 246, Springer, Berlin, 1986, pp. 82–93.
[28] W. Vogler, Failures semantics based on interval semiwords is a congruence for refinement, in: Proc. STACS'90, Lecture Notes in Computer Science, Vol. 415, Springer, Berlin, 1990, pp. 285–297.
[29] W. Vogler, Modular construction and partial order semantics of Petri nets, Lecture Notes in Computer Science, Vol. 625, Springer, Berlin, 1992.
[30] W.J. Yeh, M. Young, Compostional reachability analysis using process algebra, in: Proc. 4th Internat. Workshop on Testing and Verifications, Victoria, Canada, ACM Sigsoft, 1991, pp. 49–50.