Verification of relative correctness of Petri nets

Piotr Szwed
Katedra Automatyki, Akademia Górniczo-Hutnicza, al. Mickiewicza 30, 30-059 Kraków
pszwed@ia.agh.edu.pl

Abstract. *The paper describes a new method of requirements specification for concurrent systems modeled as Petri nets. The proposed correctness problem consists of three objects: the checked Petri net, a criterion net specifying requirements and an observation function that maps transitions in the checked model into transitions of the criterion net. The partial and the total correctness are defined and the method of their verification is proposed.*

1. Introduction. This paper presents a new method of requirements specification for concurrent systems modeled as Petri nets.

The proposed correctness problem consists of three objects: the *verified* Petri net modeling the examined concurrent system, the *criterion net* specifying the requirements and an observation function serving as the map between the verified and the criterion net.

The results described in this paper are the continuation of former works [1,2] on application of observation functions in specification and verification of concurrent systems. We use the places and transitions Petri net (PT-net) as models describing both a concurrent system and its requirements. This choice is can be justified by the similarities between matrix representation of Petri nets and linear equation models used in former papers, as well as the semantic equality between PT-nets and colored Petri nets [3], allowing to extend the range of the method application for high-level Petri nets.

We define the relative correctness problem using the linear observation function. The function maps selected transitions in verified system into sets of transitions of the criterion net. The result of the function is somehow similar to the results of application of restriction and remapping operators of CCS i CSP algebras [4, 5].

The paper is organized as follows. Chapter 2 summarizes the basic definitions concerning place and transitions Petri nets. Chapter 3 defines the linear observation function and introduces the notions of partial and total correctness for concurrent systems modeled as PT-nets. Chapter 4 describes the methods of relative correctness verification, Chapter 5 presents an example of a system implementing the alternating bit protocol and discusses the results of its verification.

2. Petri nets. Petri nets is a widely used formal tool for modeling concurrent systems due to their expression power allowing to specify easily such properties as concurrency, nondeterminism, choice, synchronization and mutual exclusion.

A Petri net is a bipartite directed graph whose vertices belong to two disjoint sets: *places* and *transitions*. A marking is an assignment of non-negative values (*tokens*) to places.

Definition 1. A Petri net is a tuple $PN = (P, T, F, W, M_0)$, where: P is a finite set of places, T is a finite set of transitions, $F \subset (P \times T) \cup (T \times P)$ is a finite set of arcs, $W : F \to \mathrm{N}$ is a function assigning weights to arcs, $M_0 : P \to \mathrm{N} \cup \{0\}$ is an initial marking. ■

A marking represents a net state. If all incoming places $\{p_i\}$ for a transition t have at least $W(p_i,t)$ tokens, the transition t is *enabled* and can be executed (*fired*). Firing a transition moves tokens from its input places to output places changing the net state (marking). Numbers of subtracted and added tokens are determined by arc weights. Subsequently executed transitions form a firing sequence $\rho = \langle t_1, t_2, \ldots, t_n \rangle$. A marking M_n is reachable from M_0 if there exist a firing sequence leading M_0 to M_n. The net is *bounded* if all markings reachable from the initial marking are bounded.

Transitions t_1 and t_2 are called *concurrent* for a marking M if both are enabled and for all their common input places the condition $M(p_i) \geq W(p_i, t_1) + W(p_i, t_2)$ holds.

In this paper we use *matrix representation* of Petri nets [6, 7]. The incidence matrix A of a Petri net has $|P|$ rows and $|T|$ columns. Its elements $A(p, t)$ specify how many tokens are removed or added to a place p as the result from firing a transition t. A firing sequence ρ leading from the marking M_0 to M_n can be described as a vector $x_n \in \mathrm{R}^{|T|}$ whose i—th element specify how many times a transition t_i fired in the sequence ρ. As a consequence we obtain the

state equation of a Petri net: $M_n = M_0 + A\,x$, where $x \in X = \mathbf{R}^{|T|}$.

3. The correctness problem. The idea of how relative correctness requirements are specified and verified can be explained taking a position of an observer who tests and debugs a system. He traces step by step executed actions (as calling a procedure, sending a message or capturing an external event). He ignores some of them, but for selected ones checks whether they are executed in an order agreed with a more abstract requirements specification. The model of requirements differs from the checked system only in the abstraction level. It can be also submitted a correctness verification in relation to a more abstract specification layer.

We model both verified and criterion specifications as Petri nets with arc weights equal to 1. The behavior of a net is represented by nondecreasing sequences of vectors $s = \langle x_1, x_2,..., x_j, x_{j+1},...,x_n,...\rangle$ called *solutions* Sequences of solutions represent a net behavior in more general form then firing sequences, because they allow to model instantaneous execution of a number of concurrent transitions in one step. We allow also multiple repetitions of the same element in a sequence. For generality, we assume, that all sequences are infinite; finite sequences can be transformed into infinite by repeating their last element.

The observation function maps sequences of solutions representing an execution of a verified net into sequences of vectors belonging to the domain of solutions of a criterion net.

The examined sequence of solutions describing a behavior of a verified net is correct relatively to a specification (meets the requirements) if its image in the observation function is a correct sequence of solutions representing transitions of the criterion net.

3.1 Linear observation function. Let us examine two Petri nets: $PN = (P, T, F, W, M_0)$ and $PN' = (P', T', F', W', M_0')$. with state equations $M = M_0 + A\,x$, where $x \in X = \mathbf{R}^{|T|}$ and $M' = M_0' + A'\,x'$, where $x' \in X' = \mathbf{R}^{|T'|}$. The first will be called a *verified net* and second a *criterion net*.

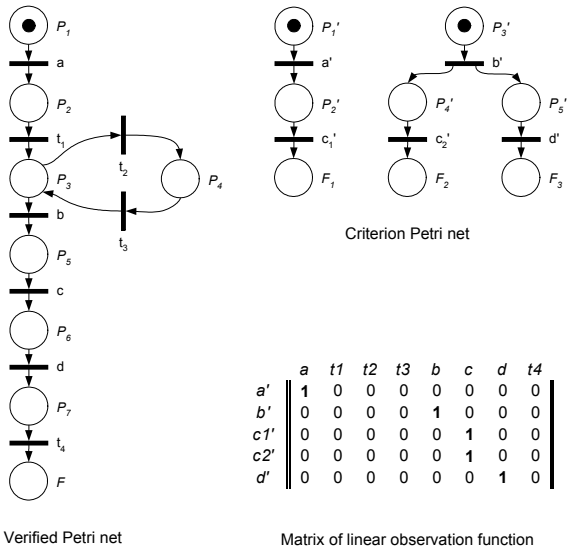
We define the function $p_\pi : T \to 2^{T'}$. It maps transitions in the verified net into sets of transitions of the criterion net. We can notice, that the image of a given transition can be an empty set (the transition is non-observable), a set containing exactly one element or a set containing a number of transitions that should be executed concurrently.

Linear observation function $\pi: X \to X'$ is described by the matrix L of size $n \times n'$, where $n = |T|$ and $n' = |T'|$. The matrix elements has values:

$$L(i,j) = \begin{cases} 1 & \text{if } t_i' \in p_\pi(t_j) \\ 0 & \text{in other case} \end{cases}$$

Solution x' satisfying $x' = \pi(x)$ is defined as: $x' = L\,x$.

The Fig. 1. shows an example of the relative correctness problem. Transitions t_1, t_2, t_3 and t_4 of the verified net *PN* are non-observable. The criterion net *PN'* specifies requirements concerning the sequence of transitions, eg. transition a' must precede c_1'. The requirements specification allows concurrent execution of transitions, eg. $\{a', b'\}$ or $\{c_1', c_2', d'\}$. In the presented example, the verified net implements the required transitions in the sequential manner.



	a	t1	t2	t3	b	c	d	t4
a'	**1**	0	0	0	0	0	0	0
b'	0	0	0	0	**1**	0	0	0
c1'	0	0	0	0	0	**1**	0	0
c2'	0	0	0	0	0	**1**	0	0
d'	0	0	0	0	0	0	**1**	0

Fig. 1 Example of a relative correctness problem

3.2. Partial correctness. Let $s = \langle x_1, x_2,..., x_i, x_{i+1},...,x_n,...\rangle$ be a sequence of solutions representing a behavior of Petri net *PN* characterized by the state equation $M = M_0 + A\,x$, $x \in X$. A vector $v_i = x_{i+1} - x_i$ is called a *step* if:

- v_i is equal to 0, what means that no observable transition occurred in the *i*-th step,
- there exists exactly one k, that $v_i(k) = 1$, what means that k–th transition was enabled at the marking $M_0 + A\,x_i$ and was executed
- there exist a set of indices K, that $v_i(k) = 1$ and all transitions t_k, for $k \in K$ were enabled at the marking $M_0 + A\,x_i$ and were concurrently executed.

We define the predicate *step*(x_i, x_j, *PN*) that evaluates to true if $x_j - x_i$ is a step at the marking $M_0 + A\,x_i$.

Let us define a predicate *dead*(x, *PN*), that evaluates to true, if the marking $M = M_0 + A\,x$ is dead. We use additional information on how the net models

underlying system to distinguish markings, that can be identified as correctly reached final states. We define a predicate *final*(x, *PN*) that is true if the marking $M = M_0 + A\,x$ is dead and only selected elements of the vector M has non-zero values. For example, for the nets in Fig. 1. the presence of a token in the place F and the absence of tokens in other places may be identified as reaching a final state; for the net *PN*' final state will correspond to marking having tokens in places F_1, F_2 and F_3.

Definition 2. Let *PN* be a verified net and *PN*' be a criterion net that are described by corresponding state equations $M = M_0 + A\,x$, where x ∈ X and $M' = M_0' + A'\,x'$, where x ∈ X'. A sequence of solutions $s = \langle x_0, x_1, x_2, \ldots, x_i, x_{i+1}, \ldots \rangle$ is correct, what we denote by a predicate *correct*(s, π, *PN*') if:

1. s is a sequence of steps for *PN*, i.e. the condition $\forall\, i \geq 0$. *step*(x_i, x_{i+1}, *PN*) holds.
2. There is no such element x_i in the sequence that *dead*(x_i, *PN*) $\wedge \neg$ *final*(x_i, *PN*)
3. Subsequent elements of the sequence $s' = \langle x_0', x_1', x_2', \ldots, x_i', x_{i+1}', \ldots \rangle$, where $x_i' = \pi(x_i)$ satisfy the predicate *step*(x_i', x_{i+1}',*PN'*) for all $i \geq 0$.
4. $\forall\, j \geq 0$. *final*(x_j, *PN*) $\Rightarrow$ *final*($\pi(x_j)$, *PN*') ■

A correct sequence of solutions should meet requirements defined by a pair (π, *PN'*). The condition states that the correct sequence is free of deadlock, is mapped by the observation function into a sequence of steps and reaching a final marking in a verified net should be accompanied by reaching a final marking in the criterion net.

Definition 3. Let us denote by S(*PN*) the set of all sequences of steps of the net *PN*. The net *PN* is *partially correct* relatively to *PN*' if the following condition holds:

$$\forall\, s \in \mathrm{S}(PN)\ .\ correct(s, \pi, PN')\ \blacksquare$$

Partial relative correctness of a net is defined as the correctness of all its sequences of steps. Such formulation refers to classical definitions of partial correctness (or more generally definitions of *safeness*) [8, 9]. Informally, this property can be express as "nothing bad can happen". In this case a bad event is making a step violating the criterion specification.

3.3 Divergence. Correctness conditions allow repeating values in the observed sequence $\langle \pi(x_0), \pi(x_1), \pi(x_2), \ldots \rangle$. The partial correctness property formulation is invariant under stuttering [10]. Models that are invariant under stuttering bring more difficulties during the verification of *liveness* properties (thus also the total correctness). The liveness properties are usually formulated as assertions, that if a certain precondition is satisfied, then after executing a final number of steps a desired event occurs, e.g. successful program termination, gaining access to a resource or activation of a component process. When we observe repeating states of a system (a sequence of identical solutions imply that the system does not change its state) we can not tell if the sequence is finite or not. Such infinite sequence can happen if the verified net executes a loop and no transition in the loop is observable.

A similar effect in CSP algebra [4] was denoted by the term *divergence*. The source of divergences is the application of restriction operator "/" that removes a set of symbols from the alphabet of a process, making transitions marked with removed actions non-observable.

The correctness problem in Fig. 1. has a divergence. It is caused by the pair of non-observable transitions t_2 and t_3, that after reaching the input place of the transition t_2 can be executed infinitely many times. From the observer's point a presence of a divergence can be considered as the lack of fairness. For a divergent system, there exist an infinite observation, for which a set of criterion net transitions is continuously enabled (in the case in Fig. 1 the enabled set contains the transition c_1'), but none of them is executed.

Definition 4. Let $s = \langle x_0, x_1, \ldots, x_i, x_{i+1}, \ldots \rangle$ be a sequence of solutions of the net *PN* described by the state equation $M = M_0 + A\,x$, and π any observation function. Observation of the sequence s has a divergence, if there is a subsequence $s_{\mathrm{div}}(x_{\mathrm{div}\,1}, x_{\mathrm{div}\,n})$ satisfying:

1. $M_0 + A\,x_{\mathrm{div}\,1} = M_0 + A\,x_{\mathrm{div}\,n}$
2. $\forall\, i : div_1 \leq i \leq div_n\ .\ \pi(x_i) = \pi(x_{\mathrm{div}\,1})$ ■

We define a predicate *divergent*(s, *PN*, π), which is true, if for a given observation function π the observation of the sequence of solutions s of the net *PN* has divergence.

3.4. Total correctness. The total correctness property imposes additional conditions on a verified net. It is required, that all transitions of the criterion net could be observed and that all observed sequences of solutions are free from divergences.

Definition 5. Let *PN* and *PN'* be Petri nets defined as in Definition 2. and π be an observation unction. The net *PN* is totally *correct in* relation to *PN*' if:

1. It is partially correct: $\forall\, s \in \mathrm{S}(PN)$. *correct*(s, π, *PN*')
2. The observation of its behavior covers the whole space X':
   $\forall\, i \leq dim$ X' . ($\exists\, s \in \mathrm{S}(PN)$. $\exists\, x_k \in s.\ x' = \pi(x_k) \wedge x'(i) > 0$), where *dim* X' denotes the size of the space X' .

3. There is no divergence: $\forall\ s \in S(PN)\ .\ \neg\ divergent(\ s, PN, \pi\ )$ ■

4. Correctness verification. Automatic correctness verification is based on the construction of the *graph of coupled net execution G* describing synchronous execution of both verified and criterion nets [1,2]. Informally, a graph G is a finite directed acyclic graph whose vertices are marked with coupled net states (M,M') and edges with pairs $(v,\pi(v))$, where $v \in X$. For an edge $(v,\pi(v))$ linking vertices (M_i,M_i') and (M_j,M_j') the equations $M_j = M_i + A\ v$ and $M_j' = M_i' + A'\ \pi(v)$ hold. The transition along an edge is correct, if v is a step in the verified net and $\pi(v)$ is a step in the criterion net.

The algorithm of correctness verification constructs a partial graph of coupled execution checking the correctness of its transitions. It can be considered as a recursive procedure realizing in-depth graph exploration. We denote by $Next(M)$ a set of transitions enabled in a marking M. Let $e(t_i)$ be a vector in X corresponding to a transition t_i; i-th element of the vector $e(t_i)$ has the value1, the other 0. Set S is a set of encountered coupled net states (M,M').

```
depthCheck (S, M, M', x){
    foreach( t ∈ Next(M) ){
        Calculate x=x+e(t); M2 = M + A·e(t) and
        M2' =M' + A'·π(e(t));
        if( ∃ i. M2'(i) < 0 )report error; /*
        sequence of solutions is not correct */
        add a vertex marked with (M2,M2') to the
        graph G ;
        add to an edge from (M,M') to (M2,M2') )
        marked with ( e(t), π(e(t)) );
        if( (M2,M2') ∈ S ) return;
        /* do not check the descendants of vertex
        that has been encountered earlier */
        Assign: S = S ∪ {(M2,M2')} ;
        depthSearch(S, M2, M2', x);
    }
}
```

The verification process starts with assignment $S=\varnothing$; $M=M_0$; $M'=M_0'$, $x=0$ and the call to `depthCheck`(S, M, M', x).

We have implemented the prototype software for relative correctness verification. Actually, the tool does not realize the presented algorithm in a recursive manner, because checking the divergence conditions requires the access to the stack of parameters. We also do not construct the full graph of coupled nets execution. The set of examined vertices and the stack of parameters of the `depthCheck` procedure provide enough information for checking the correctness conditions.

The input for the software tool is a textual specification of the verification problem. The specifications are more decorated then plain Petri net models. They contain tables of symbols assigned to transitions and places what ensures better readability of the specifications and results of their analysis. The tool produces a report on the verification process. It describes details of transitions, reached markings, sets of enabled transitions, etc.

In the case of an error, as deadlock or incorrect transition, a sequence of transitions leading to the erroneous situation is reported. It serves as the counterexample. At the termination, the tool prints the summary of verification process (number of transitions, incorrect transitions, loops, divergences).

In course of a verification a set S of reached vertices of graph G is kept in memory, as well as the current sequence of solutions starting from the initial marking. It should be remarked, that only sparse representation of vectors and matrices is used.

5. Example. As the application example we present the verification of the alternating bit protocol [5]. The parties of the protocol are two processes *Sender* and *Receiver* connected by two asynchronous communication channels *Trans* and *Ack*. The channels may represent a computer network transport mechanism.

An idea of the protocol is showed in Fig. 2. The following scenario realizes the information exchange. *Sender* after obtaining a request to transfer a message (*accept*) sends it to the channel *Trans* appending a token, being an agreed bit value (e.g. 0). *Receiver* gets the packet and if it contains the expected bit value, then delivers the message (*deliver*) and sends the acknowledgement bit to the channel *Ack*. *Sender* receives the acknowledgment bit; if its value matches the bit sent in a recent package, it is ready to accept and send the message, which will be attributed with the alternated bit value. If the acknowledgement message `ack` is lost or does not reach the *Sender* before the timeout period elapse. then the sender repeats last transmission. Acknowledgement bits are used to protect against the duplication or loss of send packets.

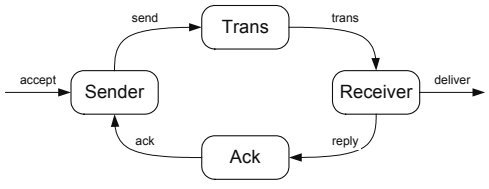


Fig. 2. Alternating bit protocol

A formal proof then the system behaves as a buffer of the capacity 1, i.e. it executes the sequence of

accept and *deliver* operations (Fig. 3.) can be found in [5]. In our case, the net modeling the buffer will serve as the criterion specification.

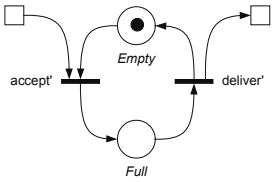


Fig. 3 Criterion Petri net specifying the requirements.

Fig. 4. shows nets modeling the processes *Sender* and *Receiver* and communication channels. For more readability we adopt the convention that transitions with the same name in different subnets (e.g..: *send(0)* in the subnet *Sender* and *send(0)* in the subnet *Trans*) refer to the same transition. It should be noticed that implicit state member of subnets *Sender* and *Receiver* are local binary variables. Their values are used to index distinguishable transitions and places (e.g..: *send(0)* and *send(1)*). The examined specification assumed that packets can be lost in communication channels (transitions *loss* in subnets *Trans* and *Ack*).

The observation function maps transitions in verified net *accept(0)* and *accept(1)* onto criterion net transition *accept* (Fig. 3) and transitions *deliver(1)* and *deliver(0)* onto *delive*

The verification proved partial correctness of the problem. The system is not totally correct because of the presence of divergences. The results of verification are summarized in the table Tab 1. Partial correctness justifies that the checked protocol model meets the requirements concerning sequence of operations (Fig. 3).

However, the presence of divergences indicates that there is no guarantee that a message after being entered (*accept*) will be output (*deliver*) or that the system in a finite time will be ready to get and send a new message. Divergences are caused by losing communication channels and transitions *timeout(0)* and *timeout(1)* of the subnet *Sender*. Revealing the divergences can be a valuable hint for a system designer. To assure the total correctness he should for example limit the number of sent packets in case of missing acknowledgements to a predefined number *n*. After sending *n* packets with identical content without acknowledgement, the process *Sender* should terminate the transmission due to communication error. Corresponding transition should be added to the criterion net defining the requirements for the protocol implementation.

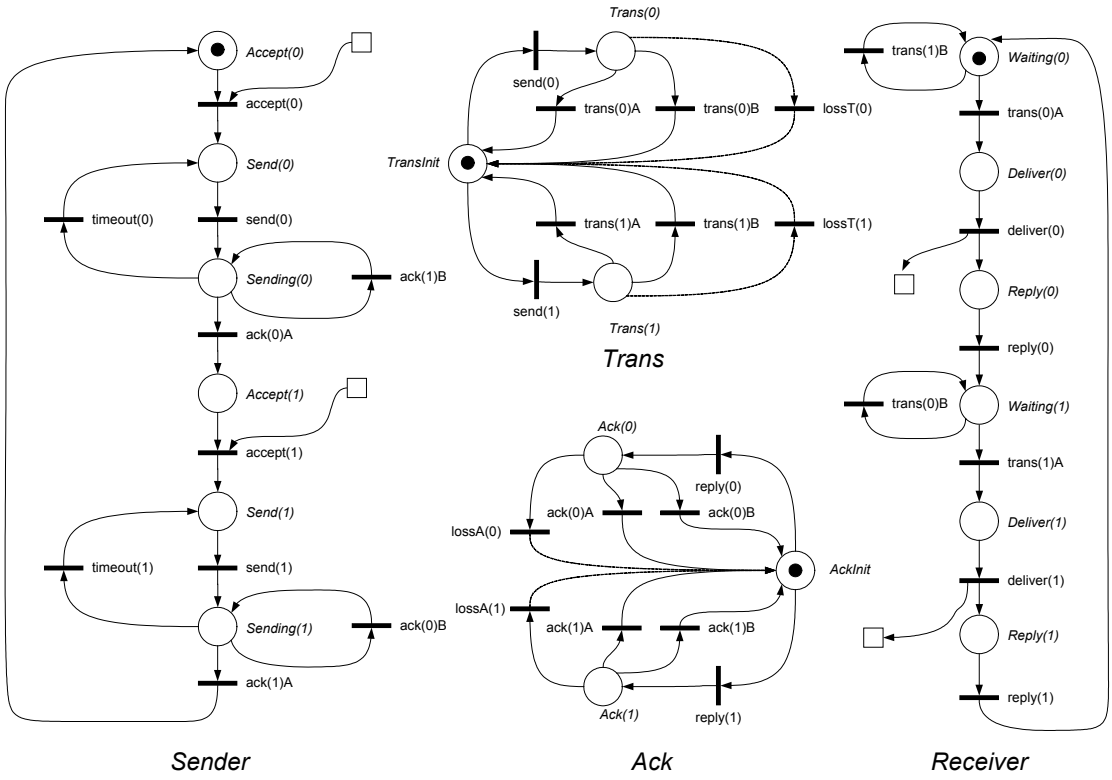


Fig. 4 Verified Petri net modeling alternating bit protocol

Tab. 1 Results of verification of the net modeling the alternating bit protocol

Number of transitions	282
Number of erroneous transitions	0
Number of vertices of coupled execution graph	116
Number of sequences of solutions leading to the branching states	71
Number of sequences of solutions with divergences	63
Number of sequences of solutions lading to deadlock markings	0

6. Conclusions. The paper describes a relative correctness problem for linear observation function. We model a concurrent system as a PT Petri net. The same formalism is used to describe the system requirements. Using the matrix representation of Petri nets we define the partial correctness as the correctness of all sequences of its steps. The satisfaction of total correctness requires additionally the lack of divergences in observations and covering all transitions in the criterion net specifying the requirements.

The example of alternating bit protocol shows a small size specification that can be effectively checked with the software tool implementing the algorithm of relative correctness verification. We have also successfully verified much more complex cases with thousands of transitions and reached states. In case of large size problems it is possible to apply partial search algorithms.

One of the most interesting candidate is *supertrace* [11]. In the supertrace algorithm the explored statespace is not represented explicitly. We represent explicitly states in the set *S* in *depthCheck* procedure in Chapter 4. The representation is compact, as we use sparse vectors, sets are implemented as trees, etc. However, the efficiency of the verification process decrease with growing number of states in the set *S*. In the supertrace algorithm states in the set are represented by bit values in a large bit vector addressed by a hash function. In this way it is possible to construct a set having millions of states. Moreover, checking if the state was encountered earlier in the search process is very fast.

The paper was supported by the KBN grant Nr *4T11C 035 24 Zastosowanie metod formalnych do wspomagania wytwarzania poprawnego oprogramowania systemów czasu rzeczywistego.*

References

[1] Szwed, P.: *Analiza poprawności oprogramowania współbieżnego z wykorzystaniem funkcji obserwacji*, praca doktorska, Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki AGH, Kraków 1999

[2] Szwed, P.: *Zastosowanie liniowej funkcji obserwacji do analizy poprawności oprogramowania współbieżnego*, Materiały VII Konferencji Systemy Czasu Rzeczywistego, Kraków 2000,Katedra Automatyki Akademii Górniczo-Hutniczej, 99–108

[3] Jensen K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use, Vol. I-III, Springer Verlag, 1995/96

[4] Hoare, C. A. R.: *Communicating Sequential Processes*, Prentice-Hall International, Englewood Cliffs, 1985

[5] Milner, R.: *Communication and Concurrency*. Prentice Hall, Englewood Cliffs, 1989

[6] Murata T.: *Petri Nets: Properties, Analysis and Applications*, Proceedings of the IEEE, Vol.77, No 4, April 1989

[7] Reisig W.: *Petri Nets – An Introduction*, EATCS Monographs on Theoretical Computer Science, Volume 4. Springer. 1985

[8] Manna, Z., Pnueli, A.: Verification of concurrent programs: The temproral framework – w BAYER. R.S., MOORE J.S. (red) *The Correctness Problem in Computer Science – International Lecture Series in Computer Science*, Academic Press, 1981, 215–272

[9] Szmuc, T.: *Poprawność współbieżnych systemów oprogramowania*, Zeszyty Naukowe AGH, Automatyka, vol. 46, 1989

[10] Lamport, L.: What good is temporal logic?, *Information Processing 83: Proc. of the 9th IFIP World Computer Congress*. Ed. R.E.A. Mason, Elsevier Publ., September 1983, 657–668

[11] Holzmann, G.J., *An improved reachability analysis technique*, Software Practice and Experience, Vol. 18, No. 2, 1988, 137–161