between adjacent face pairs is very small. We also find that the edge of each face constructed by the backprojection method always give a good individual face shape, because the small variation in the depth parameter of the face equation only affects the face shape a little. This is very useful for individual face recognition.

VII. CONCLUSION

A correspondenceless method has been presented to determine the plane equation of all visible polyhedral faces from a single view. It applies grid coding technique using two perpendicular sets of laser planes. The normal vector of each visible polyhedral face is first determined through the geometric relations and the depth parameter of the face equation is then calculated based on the given dimensions of the grid on the code plate. A sensitivity analysis indicates that the estimation is rather robust if two sufficiently long projected grid line segments are used to infer the face equation parameters. To delimit the face boundary, we use two different methods for edge construction: the backprojection method and the plane-intersection method. The goodness measures of these two method are given. Finally, the unique parameter of the face equation only affects the face shape a little.

ACKNOWLEDGMENT

The authors would like to thank the referees for constructive suggestions which lead to a better paper presentation in its current version.

REFERENCES


Representation of Nonstructured Concurrency by Petri Net Languages

Hyung Lee-Kwang and Joel Favrel

Abstract— The concurrency is classified into two types: structured concurrency and nonstructured concurrency. After showing that the nonstructured concurrency cannot be represented by the conventional notations in the Petri net language, a method to represent such concurrency by the language is proposed. The proposed method allows us to utilize the existing approaches for analyzing properties of a nonstructured concurrency by the Petri net languages.

Manuscript received June 17, 1991; revised August 28, 1992.

H. Lee-Kwang is with the Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Gungdong, Daejon, 305-701, Seoul Korea.

J. Favrel is with the Department d’Informatique, INSA de Lyon, 69621, Villeurbanne, France.

IEEE Log Number 9206193.
I. INTRODUCTION

The Petri net is a versatile modelling tool, and analysis of the petri net can reveal important properties about the structure and dynamic behavior of the modelled system. In a Petri net, actions are modelled by transitions, and sequences of actions are modelled by sequences of transitions [2]. The set of all possible sequences can be interpreted as a language in the sense of formal language theory. Many of the basic concepts of Petri net language theory have been borrowed from the classical theory of formal languages. This approach to define the language and to obtain formal properties has been used by several researchers, and there exist some approaches to analyze properties of a system by using the Petri net language [3]-[7], [10].

The Petri nets can model dynamic behavior of systems such as repetition, choice, synchronization and concurrency. The concurrency can be classified into two types: structured concurrency and nonstructured concurrency. By using the conventional notations in Petri net language, we can represent the repetition, choice, structured concurrency, etc. However, the nonstructured concurrency cannot be represented by the Petri net language in the abstract form. It is thus hard to utilize the existing approaches for analyzing properties of such concurrency. Therefore, in this paper, after characterizing the structured and nonstructured concurrency, we propose a method of representing the nonstructured concurrency by the Petri net language.

In Section I, concepts of the Petri net and its language are briefly reviewed. Section III characterizes the structured concurrency and nonstructured concurrency, and Section IV introduces a representation method of nonstructured concurrency. An amplification example is given in Section V.

II. PETRI NET LANGUAGES

In this section, we review briefly the concepts of Petri net language in following the notion and terminology in [2]. A Petri net structure, C, is a four-tuple, $C = (P, T, I, O)$. $P = \{ p_1, p_2, \ldots, p_n \}$ is a finite set of places, $n \geq 0$. $T = \{ t_1, t_2, \ldots, t_m \}$ is a finite set of transitions, $m \geq 0$. $I : T \rightarrow P^+$ is the input function, a mapping from transitions to place $O : T \rightarrow P^-$ is the output function, a mapping from transitions to places $[1]$, [2], [8], [9]. A marking $\mu$ is a function from the set of places $P$ to the nonnegative integers, $\mu : P \rightarrow N$.

A language is a set of strings over an alphabet. To consider the language of a Petri net, we must define both what the alphabet $\Sigma$ of a Petri net should be and how it is to be associated with the Petri net. In this paper, the association of symbols to transitions is made by labelling function, $\sigma : T \rightarrow \Sigma$. We limit the association to the free labelling function because it is not easy to study the sequence of transitions with the arbitrary or lambda free labelling. We define $N = (C, \sigma, \mu, F)$ to be a labeled Petri net with Petri net structure $C$, labelling $\sigma$, initial marking $\mu$, and final state set $F$. For labeled Petri net $N$, language $L(N)$ can be defined [2].

Petri nets represent system behaviors such as sequence, repetition, choice, and concurrency. To represent the system behaviors in Petri net language (L), the operators $\cdot$, $\circ$, $\ast$, $\parallel$, $\mathrm{random}$ and parentheses are used as follows [2], [11]:

1. The operation $\cdot$ in $L_1 \cdot L_2$ (or $L_1 \cdot L_2$) is the concatenation of elements of $L_1$ and $L_2$ such as $L_1 \cdot L_2 = \{ a \cdot b | a \in L_1, b \in L_2 \}$.
2. The operation $\ast$ in $L_1 + L_2$ denotes the union (choice) of the sets $L_1$ and $L_2$, which is $L_1 + L_2 = \{ a + b | a \in L_1, b \in L_2 \}$.
3. The operator $\mathrm{random}$ or iterate of a set is defined as the infinite union, i.e., $a^* = \lambda + a + aa + aaa + \cdots$.
4. The operation $//)$ indicates the concurrency of the set $L_1$ and $L_2$ such as $L_1 // L_2 = \{ a // b | a \in L_1, b \in L_2 \}$.

III. CHARACTERIZATION OF STRUCTURED AND NONSTRUCTURED CONCURRENCY

The transition in Petri nets can be classified into two types as follows:

Distribution transition: A transition that has more than one input place as shown in Fig. 1.

Synchronization transition: A transition with more than one output place as shown in Fig. 2.

A transition that has more than one output place and more than one input place can be a distribution and synchronization simultaneously. A subnet is a net involved in a Petri net, and the exterior of the subnet is the part that is not involved in the subnet [8].

To establish the concept of structured and nonstructured concurrency in Petri nets, it is necessary to define the term of frontier node of a subnet.

1. Frontier node: A node that is connected to the exterior of a subnet. Input frontier node is connected by an incoming arc and output frontier node by an outgoing arc.

Now we define the structured and nonstructured concurrency as follows:

Structured concurrency: concurrency represented by a subnet (structured net) that verifies the following properties (see Fig. 3):
Nonstructured Concurrency: Concurrency represented by a subnet (nonstructured net) that does not verify the above properties (see Fig. 4). If one property is violated, the concurrency is nonstructured.

Let’s consider the nonstructured concurrency in Fig. 4. From this concurrency, we can obtain five firing sequences of transitions:

\[\begin{align*}
\text{a) } & \quad \text{b) } \quad \text{c) } \\
\text{abcde} & \quad \text{abcdef} & \quad \text{acbed} & \quad \text{acbedf} & \quad \text{abcd} \\
\text{ef} & \quad \text{ef} & \quad \text{ef} & \quad \text{ef}
\end{align*}\]

To represent the previous sequences by a Petri net language, we can consider \(a(b//c//d//e//f)\). This language represents the previous sequences (1)-(4), but not sequence (5). Now we can state that we can not represent nonstructured concurrency by Petri net language in abstract form.

IV. REPRESENTATION OF NONSTRUCTURED CONCURRENCY

To represent the nonstructured concurrency, we propose a notation called pre-exponential notation that represents a constraint on a sequence. The notation is given as “\(\cdot\)”. It indicates that transition \(c\) is fired after the firing of transition \(a\). That is, this notation represents a constraint on the sequence between \(a\) and \(c\). For example, Petri net language \(ab//c\) can give sequences \(a\ b\ c\) and \(a\ c\ b\). Compare with \(ab//c\), which gives sequences \(a\ b\ c\) and \(a\ b\ c\).

Now, with the pre-exponential notation, we can represent the nonstructured concurrency in Fig. 4 such as \(L = (a(b//c//d//e//f))\).

The procedure of generation of Petri net language representing concurrency (structured or nonstructured) can be summarized as follows:

Input : a Petri net
Output : Petri net language

Begin : Search distribution transition.
If the transition is found, then
   define subnet \(N_{i-1}\) whose node is the found distribution transition,
   \(\text{PARALLEL} \neq \text{empty}\).
Else
   \(\text{PARALLEL} = \text{empty}\).
While \(\text{PARALLEL} \neq \text{empty}\), do.
   Begin \(i^{th}\) iteration.
   Find the first synchronization transition from the subnet \(N_{i-1}\).
   If the transition is found,
       define subnet \(N_i\), composed...
system. We can see that our proposed method allows us to apply the existing approaches to such concurrency for analyzing properties such as the hierarchy of languages and the decomposition of languages [4]-[7], [10].

V. CONCLUSION

The concurrency represented by a Petri net has been classified into two types: structured and nonstructured concurrency. It has been pointed out that it is hard to analyze the nonstructured concurrency by using the Petri net languages because we can not represent such concurrency by the languages. We have proposed a method to represent the nonstructured concurrency by the languages in the abstract form, and developed a procedure to generate the languages to represent the concurrency. The proposed method can allow us to utilize the existing approaches for analyzing properties of the nonstructured concurrency. The proposed notation can be used to represent the sequences in graphs and networks.

Fig. 6. Partition tree of the system in Fig. 5.

REFERENCES