

Generalized Net Representation of Dataflow Process Networks

Dimitar G. Dimitrov

Abstract This paper presents translation rules for mapping from a given dataflow process network (DPN) to a generalized net (GN). The so obtained GN has the same behaviour as the corresponding DPN. A reduced GN that represents the functioning and the results of the work of an arbitrary DPN is also defined.

Keywords Generalized nets • Dataflow process networks • Kahn process networks

1 Introduction

Generalized Nets (GNs) are defined as extensions of ordinary Petri nets, as well as of other Petri nets modifications [1]. The additional components in GN definition give more and greater modeling possibilities and determine the place of GNs among the separate types of Petri nets, similar to the place of the Turing machine among finite automata. GNs can describe wide variety of modeling tools such as Petri nets and their extensions [1, 2], Unified modeling language (UML) [7], Kahn Process Networks (KPN) [4], etc. In this paper we shall show how GNs can adequately represent dataflow process networks (DPN).

Dataflow process networks are a model of computation (MoC) used in digital signal processing software environments and in other contexts. DPN are a special case of Kahn Process Networks [5, 6].

The structure of this paper is as follows. In next section a brief introduction of DPN is presented. In Sect. 3 a procedure for translating a concrete DPN to a GN is given. In the next section, a universal GN that represents the functioning and the

This work is supported by the National Science Fund Grant DID 02/29 “Modelling of Processes with Fixed Development Rules (ModProFix)”.

D.G. Dimitrov (✉)

Faculty of Mathematics and Informatics, Sofia University,
5 James Bourchier Blvd., Sofia, Bulgaria
e-mail: dgdimetrov@fmi.uni-sofia.bg

results of the work, an arbitrary DPN is formally defined. After that the advantages of using GN instead of DPN are discussed. Finally, future work on this subject is proposed, as well as software implementation details are given.

2 Dataflow Process Networks

DPNs consist of a set of data processing nodes named actors which communicate through unidirectional unbounded FIFO buffers. Each actor is associated with a set of firing rules that specify what tokens must be available in its inputs for the actor to fire. When an actor fires, it consumes tokens from its input channels and produces tokens in its output channels [8, 9]. Figure 1 shows a DPN with four actors.

A (*dataflow*) *actor* with m inputs and n outputs is a pair $\langle R, f \rangle$, where

$$R = \{R_1, R_2, \dots, R_k\}$$

is a set of *firing rules*. Each firing rule constitutes a set of patterns, one for each input:

$$R_i = \{R_{i,1}, R_{i,2}, \dots, R_{i,m}\}$$

A pattern $R_{i,j}$ is a finite sequence of data elements from j -th channel's alphabet. A firing rule R_i is satisfied, iff for each $j \in \{1, 2, \dots, m\}$ $R_{i,j}$ forms a prefix of the sequence of unconsumed tokens at j -th input. An actor with $k = 0$ firing rules is always enabled. $R_{i,j} = \perp$ denotes that any available sequence of tokens is acceptable from input j . “*” denotes a token wildcard (i.e., any value is acceptable).

$f : S^m \rightarrow S^n$ is a function that calculates the token sequences that are to be output by the actor, where S^m is the set of all m -tuples of token sequences.

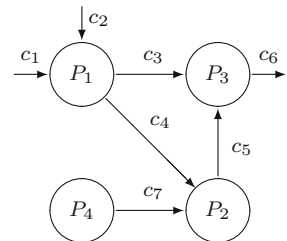
DPNs are an untimed model. Firing rules are not bounded to a specific time moment.

DPNs do not over specify an execution algorithm. There are many different execution models with different strengths and weaknesses.

To avoid confusion with tokens in GNs, in this paper we shall denote channel tokens in DPN as *data elements*.

With $pr_i A$ we shall denote the i -th projection of the n -dimensional set A where $n \in \mathbb{N}$, $1 \leq i \leq n$.

Fig. 1 Example DPN



3 Translating a DPN to a GN

In generalized nets, as in Petri nets, transitions represent discrete events, and places represent conditions (either preconditions or postconditions, depending on arc directions). Dataflow actors consume tokens from its input channels and produce tokens to its output channels. Similarly when a GN transition is activated, it transfers tokens from its input places to its output places. Dataflow actors fire when tokens in the inputs satisfy given conditions (firing rules). The same way token transfer in GN occurs when predicates associated with transitions are evaluated as true. Thus actors in DPN can obviously be mapped to GN transitions and firing rules can be seen as a special case of GN transition predicates. GN transitions' concept of firing rules—transition types—will not be used, because they ensure only the presence of a token in a given input, not its contents.

In GN, places have characteristic functions. They calculate the new characteristics of every token that enters an output place. This can be used as analog of DPN actors' firing function that calculates output sequences based on actors' input data. Channels in DPN which connect dataflow actors can be translated to GN places. Similar approach of mapping from dataflow actors to Petri net transitions and channels to places is also used in [10]. DPN tokens can be directly mapped to GN tokens but in order to preserve their ordering, whole data sequence can be represented as a single GN token.

Table 1 summarizes the translation rules from DPN to GN.

Figure 2 shows a GN representation of the example DPN in Fig. 1.

Token splitting and merging must be enabled for the GN, e.g., operator $D_{2,2}$ must be defined for it [2].

In this paper we shall use the following function to check whether an actor can fire:

$$c(R, I) = (\exists i \in \{1, \dots, |R|\} : \forall R_{ij} \in R_i : R_{ij} \subseteq I_j)$$

where R is a set of firing rules, I is a list of data element sequences, one for each input channel, and $p \subseteq q$ is true iff the sequence p is a prefix of q .

Each dataflow actor $\langle R, f \rangle$ can be translated to a transition in the following form:

$$Z = \langle \{l'_1, \dots, l'_m, l, l^*\}, \{l''_1, \dots, l''_n, l, l^*\}, *, *, r, *, * \rangle$$

Table 1 Mapping from dataflow process networks to generalized nets

	DPN	GN
Actor	Node	Transition
Channel	Arc	Place
Firing rules	(First component of an actor)	Transition predicate
Firing function	(Second component of an actor)	Characteristic function
Data	Tokens with single data elements	Token with one or more data elements

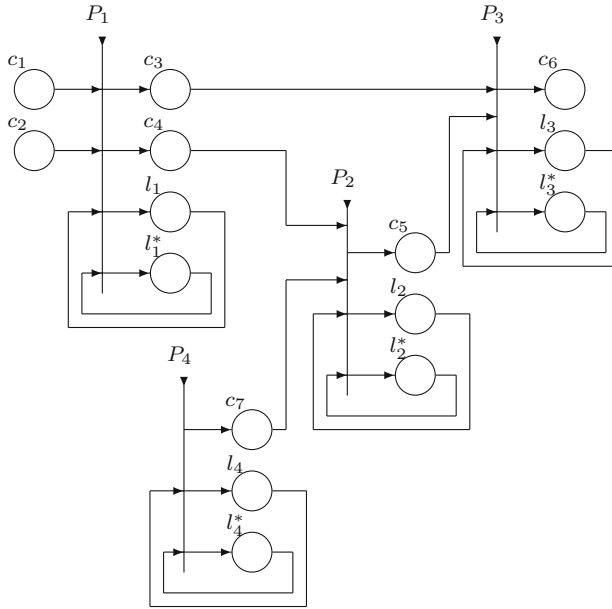


Fig. 2 GN representation of the example GN

where

- l'_1, \dots, l'_m are places that correspond to the actor's inputs;
- l''_1, \dots, l''_n are places that correspond to the actor's outputs;
- l is a place that collects all input data consumed by one firing of the actor and l 's characteristic function calculates the actor's output sequences;
- l^* is a place that keeps all input data before it can be consumed;
- tokens in l^* are merged into token δ with the following characteristic:

$$x_{s+1}^\delta = \{ \langle l_i, get(x_s^\delta, l_i) \cdot x_{s+1}^{\delta_{l_i}} \rangle | l_i \in \{l'_1, \dots, l'_m\} \}$$

where δ_{l_i} is the token (if such available) in the input place l_i , x_s^δ is the previous characteristic of token δ which loops in l^* (if available, empty set otherwise), “.” is sequence concatenation and get is a function that gets the content of a given channel stored as characteristic in δ ;

- r is the following index matrix (IM):

$$r = \begin{array}{c|cccc} & l''_1 & \dots & l''_n & l^* & l \\ \hline l'_1 & & & & & \\ \vdots & & & & & \\ l'_m & & & & & \\ l^* & & & & & \\ l & & & & & \end{array}$$

$\begin{matrix} & l''_1 & \dots & l''_n & l^* & l \\ \hline l'_1 & & & & & \\ \vdots & & & & & \\ l'_m & & & & & \\ l^* & & & & & \\ l & & & & & \end{matrix}$

where predicate $W_{l^*,l}$ checks whether exists a firing rule from R that is satisfied by current input:

$$W_{l^*,l} = c(R, pr_2 x^\delta)$$

- the characteristic function Φ is defined for l as follows:

$$\Phi_l = \{\langle l_j'', pr_j f(pr_1 x^\delta) \rangle | 1 \leq j \leq n\}$$

where f is the function associated with the dataflow actor corresponding to Z . As a side effect Φ_l removes consumed data from the characteristic of the token in l^* (which may result in empty characteristic of this token, if there is no unconsumed data);

- Φ for outputs l_1'', \dots, l_n'' retains only the data sequence which corresponds to the given output place (previously the token contains information for all output channels):

$$\Phi_{l_j''} = get(x^\delta, l_j''), 1 \leq j \leq n$$

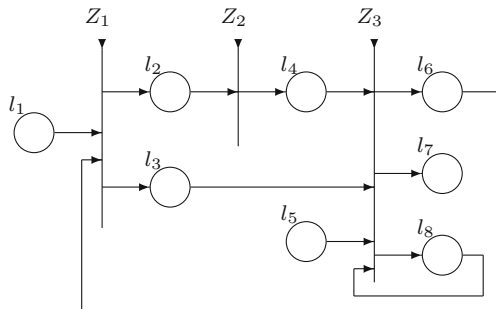
A special case are dataflow actors without inputs. Such actors are always enabled [9]. The so defined GN transition Z is capable of representing such actors. An empty token should always be present in l^* meaning that there is no input data. Predicate $W_{l^*,l}$ is always evaluated as true.

4 Universal GN for Dataflow Process Networks

Below is a formal definition of a reduced GN E that represents any DPN N . The graphical structure of E is shown in Fig. 3. Token splitting and merging must be enabled for the net.

$$E = \langle \langle \{Z_1, Z_2\}, *, *, *, *, *, *, * \rangle, \langle \{\alpha\}, *, *, * \rangle, \langle *, *, *, * \rangle, \langle \{x_0^\alpha\}, *, *, * \rangle, \Phi, * \rangle$$

Fig. 3 Graphical structure of the universal GN representing dataflow process networks



The first transition is responsible for dividing dataflow actors into two sets—ones that can be fired, according to their firing rules, and ones, that cannot:

$$Z_1 = \langle \{l_1, l_6\}, \{l_2, l_3\}, *, *, r_1, *, * \rangle$$

Token α enters place l_1 containing as a characteristic information about the DPN in the following form:

$$x^\alpha = \{ \langle \langle R, f \rangle, \{ \langle i_1, s_{i_1} \rangle, \dots, \langle i_{m_k}, s_{i_{m_k}} \rangle \} \rangle \}$$

where $\langle R, f \rangle$ denotes a dataflow actor and s_{i_j} is a sequence of initial data elements in j -th input of the actor.

A token may be available in either l_1 or l_6 . There is a token in l_1 only in the first step of the functioning of E .

Token α is transferred to l_2 if it contains at least one actor that can be fired. If there is at least one actor whose firing rules are not satisfied by input data, the token splits and goes to l_3 too. The predicates of Z_1 are the following:

$$r_1 = \frac{l_2 \quad l_3}{l_1 \begin{array}{|c|} W_1 \\ W_2 \end{array} \quad l_6 \begin{array}{|c|} W_1 \\ W_2 \end{array}}$$

where $W_1 = (\exists a \in x^\alpha : c(pr_1 pr_1 a, pr_2 pr_2 a))$ and $W_2 = (\exists a \in x^\alpha : \neg c(pr_1 pr_1 a, pr_2 pr_2 a))$

The characteristic function for l_2 and l_3 retains only firable and non-firable actors, respectively:

$$\Phi_{l_2} = \{ \langle \langle R, f \rangle, \{ \langle i_1, s_{i_1} \rangle, \dots, \langle i_{m_k}, s_{i_{m_k}} \rangle \} \rangle | c(R, \{i_1, \dots, i_{m_k}\}) \}$$

$$\Phi_{l_3} = \{ \langle \langle R, f \rangle, \{ \langle i_1, s_{i_1} \rangle, \dots, \langle i_{m_k}, s_{i_{m_k}} \rangle \} \rangle | \neg c(R, \{i_1, \dots, i_{m_k}\}) \}$$

After passing the second transition, actors are fired and their firing function f is calculated. Data elements are read from inputs and output data is written to outputs.

$$Z_2 = \langle \{l_2\}, \{l_4\}, *, *, *, *, * \rangle$$

The characteristic function for Φ_{l_4} calls the firing functions of each actor in α_1 , removes consumed data elements and stores output sequences in a new characteristic named *output_data*:

$$output_data = \{ \langle o_{i,j}, pr.f_i(I_i) | 1 \leq i \leq |x^\alpha| \rangle \}$$

where $o_{i,j}$ is j -th output of i -th actor, f_i is the firing function of i -th actor and I_i are its inputs.

The last transition Z_3 merges actors in a single token α again. It also collects input data from input channels (which is not generated by an actor from the DPN).

$$Z_3 = \langle \{l_3, l_4, l_5\}, \{l_6, l_7, l_8\}, *, *, r_3, *, \square_2 \rangle$$

Tokens carrying data elements corresponding to input channels enter in l_5 . Characteristics have the following form:

$$x^\delta = \{\langle c, s \rangle\}$$

where c is a channel and s is a sequence of data elements.

Z_3 has the following predicate matrix:

$$r_3 = \begin{array}{c|ccc} & l_6 & l_7 & l_8 \\ \hline l_3 & W & false & false \\ l_4 & W & true & false \\ l_5 & false & false & true \\ l_8 & W & false & \neg W \end{array}$$

Predicate W checks whether tokens exist in both l_3 and l_4 . If there is incoming data but no actor tokens, input data is collected in l_8 and waits.

In order for input data and actor information to be merged easily, all actors must be available:

$$\square_2 = \vee(\wedge(l_3, l_4), l_5, l_8)$$

Tokens in place l_6 are merged and the new characteristic is the union of the characteristics of α_1 and α_2 . The characteristic of the token from l_8 (if such is available) is merged with the characteristic named *output_data* (the two sets do not intersect). Merging is executed before the calculation of Φ_{l_6} .

In l_8 tokens are merged. The new characteristic is calculated in similar way as in place l^* from Sect. 3:

$$x_{s+1}^\delta = \{\langle i, get(x^\delta, i).get(x^{\delta'}, i) \rangle | i \in pr_1 x^\delta \cup pr_1 x^{\delta'}\}$$

where δ denotes the token from l_8 and δ' the new token coming from l_5 .

In l_6 after tokens are merged, the characteristic function writes actors' output data into the corresponding channels. After that it removes the *output_data* characteristic from α :

$$\Phi_{l_6} = \{\langle \langle R, f \rangle, \{ \langle i_j, s_{i_j}.get(output_data, i_j) \rangle | 1 \leq j \leq m_k \} \rangle\}$$

Data written to output channels (which do not act as input channels for other actors) leaves the net through place l_7 . The new characteristic that tokens receive in l_7 consists of only output data written to such channels (previously tokens contains output data for all channels).

$$\Phi_{l_7} = \{s \in output_data | pr_1 s \notin \bigcup pr_1 pr_2 x^\alpha\}$$

5 Conclusion

The so constructed in this paper GNs are reduced ones, i.e., this work shows that even simple class of GNs is capable of describing DPNs. If one uses a GN to model a real process, usually modeled by a DPN, he will receive many advantages, since GNs are more detailed modeling tool. First, token characteristics in GNs have history, so in the universal GN all data elements that pass through a given channel are remembered. Complex dataflow actors' mapping functions by default are translated to characteristic functions but they can also be represented as GNs. If a real process that runs in a given time is represented by a GN instead of a DPN, the modeler can use the global time component of the GN, so process time can be mapped to the time scale. Unlike DPNs and KPNs, GNs support different types of time. As in Kahn process networks' universal GN [4], a scheduler that manages the execution order of actors and channel capacities can easily be integrated.

As a future work on the topic we can define GNs for other models of computation, as well as for some special cases of DPN such as synchronous DPN.

GN IDE [3], the software environment for modeling and simulation with GNs, can be extended to support different modeling instruments such as KPNs, DPNs, as well as Petri nets, and their various extensions. The results of current research imply that the above functionality can easily be implemented without modifying GNTicker—the software interpreter for GNs, used by GN IDE.

Another potential direction is to introduce fuzzyness in DPN. Several fuzzy GN extensions are defined [2] and can be used to model such fuzzy DPNs.

References

1. Atanasov, K.: Generalized Nets. World Scientific, Singapore, New Jersey, London (1991)
2. Atanasov, K.: On Generalized Nets Theory. Prof. Marin Drinov Academic Publishing House, Sofia (2007)
3. Dimitrov, D.G.: GN IDE—a software tool for simulation with generalized nets. In: Proceedings of Tenth International Workshop on Generalized Nets, pp. 70–75. Sofia, 5 Dec 2009
4. Dimitrov, D.G., Marinov, M.: On the representation of Kahn process networks by a generalized net. In: 6th IEEE International Conference on Intelligent Systems (IS'12), pp. 168–172. Sofia, Bulgaria, 6–8 Sept 2012
5. Geilen, M., Basten, T.: Requirements on the execution of kahn process networks, In: Degano, P. (eds.) Programming Languages and Systems. In: 12th European Symposium on Programming, ESOP 2003, Proceedings, pp. 319–334. Warsaw, Poland, 7–11, LNCS 2618. Springer, Berlin, Germany (2003)
6. Kahn, G.: The semantics of a simple language for parallel programming. In: Rosenfeld, J.L. (ed.) Information Processing 74. In: Proceedings of IFIP Congress 74, North-Holland, Stockholm, Sweden, 5–10 Aug 1974

7. Koycheva E., Trifonov, T., Aladjov, H.: Modelling of UML sequence diagrams with generalized nets. In: International IEEE Symposium on Intelligent Systems, (79–84), Varna, IEEE (2002)
8. Lee, E., Matsikoudis, E.: The semantics of dataflow with firing. In: Huet, G., Plotkin, G., Lvy, J.-J., Bertot, Y. (eds.) Chapter in From Semantics to Computer Science: Essays in Memory of Gilles Kahn, Preprint Version, 07 March 2008, Copyright (c) Cambridge University Press
9. Lee, E., Parks, T.: Dataflow process networks. Readings in Hardware/Software Co-design. Kluwer Academic Publishers Norwell, pp. 59–85 (2002). ISBN:1-55860-702-1
10. Rocha, J.-I., Gomes, L., Dias, O.P.: Dataflow model property verification using Petri net translation techniques. INDIN 2011, pp. 783–788, 26–29 July 2011

Recent Contributions in Intelligent Systems

Sgurev, V.; Yager, R.R.; Kacprzyk, J.; Atanassov, K.T.

(Eds.)

2017, X, 390 p. 92 illus., 23 illus. in color., Hardcover

ISBN: 978-3-319-41437-9