On Verification of Marking-Dependent Terminacy for Data-Flow Program Nets

Shingo Yamaguchi[1], Keisuke Komiya[1], Qi-Wei Ge[2] and Minoru Tanaka[1]
[1]Graduate School of Science and Engineering, Yamaguchi University
16-1 Tokiwadai 2-chome, Ube 755-8611, Japan
[2]Faculty of Education, Yamaguchi University
1677-1 Yoshida, Yamaguchi 753-8513, Japan
E-mail : {shingo, m009vk, gqw, tanakam}@yamaguchi-u.ac.jp

Abstract: In this paper, we discuss terminacy at the initial marking for (data-flow) program nets. Ge et al. have proposed an algorithm to verify terminacy at any marking, i.e. structurally terminacy. However, there is no algorithm to verify terminacy at the initial marking. In this paper, we give a necessary and sufficient condition to verify the terminacy. Using this condition, we construct a polynomial time algorithm for a subclass of program nets. We also show that the problem to verify the terminacy for general program nets is undecidable.

1. Introduction

(Data-flow) program nets has been proposed for representing and analyzing a data-flow programs [1]. A program net must always be terminating for the following reasons: (i) A program net that isn't terminating may require unbounded resources, and (ii) the program net cannot output any computation results.

Ge et al. have proposed an algorithm to verify terminacy at any marking, i.e. *structural terminacy* [2] . This algorithm, however, cannot be applied to the verification of terminacy at the initial marking, i.e. *marking-dependent terminacy*. As such an example, let us consider a program net PN_1 with the initial marking $\mathbf{d}^0$, which is shown in Fig. 1 (a). PN_1 is terminating at the initial marking $\mathbf{d}^0$. If PN_1 has a marking $\mathbf{d}_1$ shown in Fig. 1 (b), PN_1 is not terminating at making $\mathbf{d}_1$. Therefore PN_1 is not structurally terminating but is marking-dependent terminating. We need an algorithm to verify marking-dependent terminacy, but the algorithm has not been proposed.

This paper deals with marking-dependent terminacy for program nets. In this paper, we give a necessary and sufficient condition to verify the terminacy. Using this condition, we construct a polynomial time algorithm for a subclass of program nets. We also discuss computation complexity of the problem to verify the terminacy for general program nets.

2. Preliminary

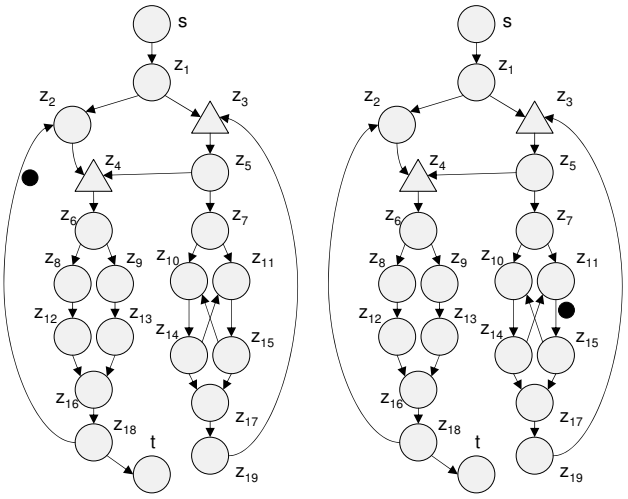
This section presents the definition and basic properties of program nets.

2.1 Program Nets

The formal definition of program nets (of canonical form [3]) is given as follows.

Definition 1: (program nets [1]) A program net is a pair $PN = (N, E)$, where

- N is a set of nodes. A node $n \in N$ is one of three types: AND-node (◯), OR-node (△), SWITCH-node (▽°). There is a single source AND-node, called start node s, and a single sink AND-node, called termination node t. Each node is located on a path from s to t.
- E is a set of directed edges (edges for short). An edge $e \in E$ is one of two types: dataflow-edge (→) and controlflow-edge (⇢). Each node has at most three dataflow-edges. □



(a) The initial marking $\mathbf{d}^0$. At this marking, PN_1 is terminating.

(b) A marking $\mathbf{d}_1$. At this marking, PN_1 is not terminating.

Figure 1. A program net PN_1.

Program nets without SWITCH-node are called SWITCH-less nets.

Figure 2 shows an example of program nets. The net calculates the factorial of an integer n, i.e. $n!$.

A token (•) represents single datum. A marking $\mathbf{d}$ expresses number of token on each edge as $|E|$-dimension vector, and the initial marking is denoted by $\mathbf{d}^0$. A program net PN with a marking $\mathbf{d}$ denoted by $(PN, \mathbf{d})$. The marking is changed according to the (node) firing rules shown in Table1. We assume that start node s fires only once in $\mathbf{d}^0$.

2.2 Terminacy

We give definitions on marking-dependent terminacy. In the remaining of this paper, "the terminacy" means marking-dependent terminacy.

Definition 2: (Eventually Dead [2]) Let $\mathbf{d}$ be a marking. A node z is said to be $\mathbf{d}$-Edead ($\mathbf{d}$-eventually dead) iff $\exists K$: z cannot fire after firing K times.

□

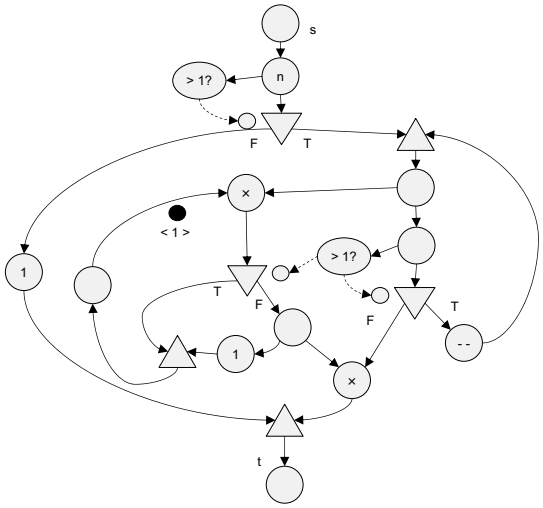
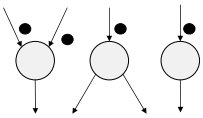
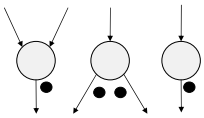
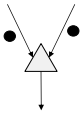
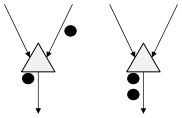
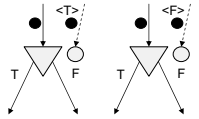
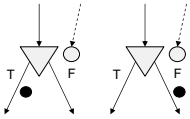


Figure 2. An example of program net. The net calculates factorial $n!$.

Table 1. Firing rule in program nets

Node Type	Before Firing	After Firing
AND		
OR		
SWITCH		

Definition 3: (Terminacy [2]) A program net $(PN, \mathbf{d}^0)$ is said to be *terminating* iff all nodes in PN are $\mathbf{d}^0$-Edead. □

We give properties on terminacy as follows.

Definition 4: (Structurally Edead [2]) A node z is said to be structurally Edead iff and z is **d**-Edead at any marking **d**. □

Definition 5: (Structural Terminacy [2]) A program net PN is said to be *structurally terminating* iff all node in PN are structurally Edead. □

Definition 6: (Dead [4]) Let Σ be set of all firing sequences in a program net $(PN, \mathbf{d}^0)$. A node z is said to be $\mathbf{d}^0$-dead if z satisfies $\forall \sigma \in \Sigma : \bar{\sigma}(z) = 0$. A program net PN is said to be $\mathbf{d}^0$-partially dead if PN satisfies $\exists z \in N, \forall \sigma \in \Sigma : \bar{\sigma}(z) = 0$. □

3. Criterion and Verification Algorithm of Marking-Dependent Terminacy

In this section, we give a necessary and sufficient condition to verify the terminacy. Next, we propose an algorithm to verify the terminacy for SWITCH-less nets according to this condition.

3.1 Criterion of Marking-Dependent Terminacy

The terminacy can be verified by investigating whether all nodes in a program net are $\mathbf{d}^0$-Edead. A sufficient condition to a node be $\mathbf{d}^0$-Edead is given as follows.

Property 1: Let $\mathbf{d}^0$ is the initial marking of a program net.

- An AND node is $\mathbf{d}^0$-Edead if at least one of its input nodes is $\mathbf{d}^0$-Edead.
- An OR-node is $\mathbf{d}^0$-Edead if all of its input nodes are $\mathbf{d}^0$-Edead.
- A SWITCH-node is $\mathbf{d}^0$-Edead if at least one of its input nodes is $\mathbf{d}^0$-Edead. □

According to this property, we give a necessary and sufficient condition to verify whether all nodes in a program net are Edead as follows.

Theorem 1: All nodes in program net $(PN, \mathbf{d}^0)$ are $\mathbf{d}^0$-Edead iff it satisfies at least one of following conditions:

(i) There is no cycle.

(ii) On each cycle, there is at least one node z such that one of the following conditions.

- z is reachable from a $\mathbf{d}^0$-Edead node by a path consisting of AND-nodes and SWITCH-nodes.
- z is $\mathbf{d}^0$-dead. □

According to Definition 3, we give a necessary and sufficient condition to verify whether a program net is terminating as follows.

Corollary 1: A program net $(PN, \mathbf{d}^0)$ is $\mathbf{d}^0$-terminating iff it satisfies one of conditions in Theorem 1. □

3.2 Verification Algorithm of Marking-Dependent Terminacy for SWITCH-Less Nets

For SWITCH-less nets, each condition in Property 1 is also necessary.

Property 2: Let $\mathbf{d}^0$ is the initial marking of a program net.

- An AND node is $\mathbf{d}^0$-Edead iff at least one of its input nodes is $\mathbf{d}^0$-Edead.
- An OR-node is $\mathbf{d}^0$-Edead iff all of its input nodes are $\mathbf{d}^0$-Edead. □

According to Corollary 1, we construct an algorithm to verify the terminacy for SWITCH-less nets based on DFS (Depth-First Search). This algorithm searches $\mathbf{d}^0$-Edead nodes from start node s and delete them. It judges the program net terminating if it can delete all nodes. It searches nodes on the basis of Property 2.

≪ Verification of Marking-Dependent Terminacy ≫

Input : SWITCH-less net $PN_{SWL} = (N, E)$, initial marking $\mathbf{d}^0$

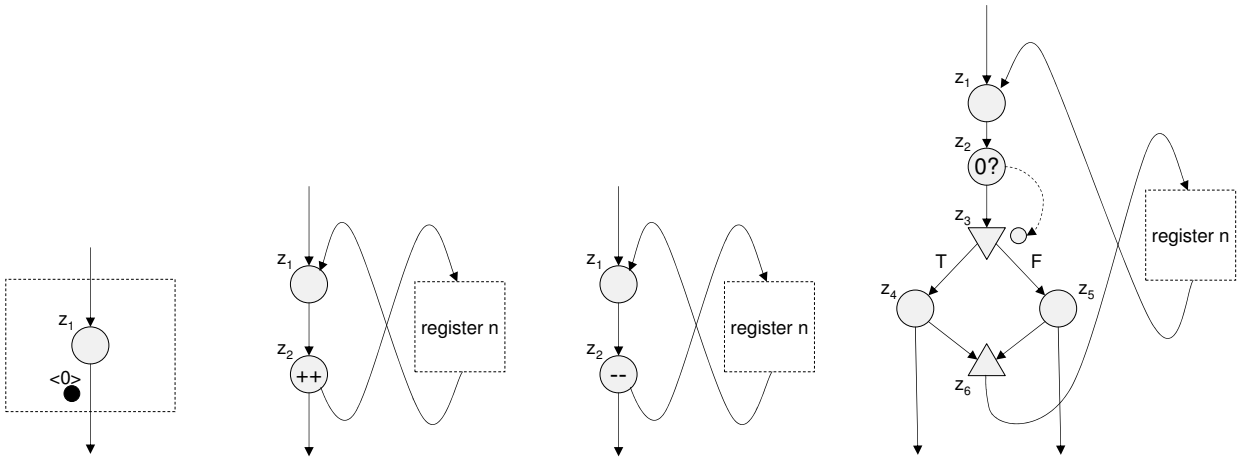
Output : Is PN_{SWL} terminating at $\mathbf{d}^0$?

CHECK-TERMINACY(PN_{SWL}, $\mathbf{d}^0$)

1° Delete all $\mathbf{d}^0$-dead nodes. Use ≪ Find $\mathbf{d}^0$-Dead Nodes ≫ (see Appendix) to find $\mathbf{d}^0$-dead nodes.

2° Rewrite each OR-node z_{OR} such that $|\bullet z_{OR}| = 1$ as an AND-node.

3° Execute DELETE-ANDNODEPATH(s), where s is start node.



(a) register n (b) instruction $I(n)$ (c) instruction $D(n)$ (d) instruction $J(n)[s]$

Figure 3. Program nets representing the instructions for a register machine.

4° Rewrite each OR-node z_{OR} such that $|\bullet z_{OR}| = 1$ as an AND-node.

5° Without an OR-node z_{OR} such that $\bullet z_{OR} = \phi$, go to 8°.

6° Execute DELETE-ANDNODEPATH(z_{OR}) for all z_{OR}, where z_{OR} is an OR-node such that $\bullet z_{OR} = \phi$.

7° Go to 4°.

8° If $N = \phi$ then return true, else return false.

DELETE-ANDNODEPATH(z)

1° Mark z as a visited node.

2° if z is not visited then execute DELETE-ANDNODEPATH(z_{AND}), where z_{AND} is an AND-node such that $z_{AND} \in \bullet z$.

3° Delete z.

We can obtain the following theorem about the computation complexity of ≪ Verification of Marking-Dependent Terminacy ≫, because this algorithm is based on DFS.

Theorem 2: ≪ Verification of Marking-Dependent Terminacy ≫ runs in polynomial time. □

4. Computation Complexity of Marking-Dependent Terminacy

≪ Verification of Marking-Dependent Terminacy ≫ enables us to verify the termiacy for SWITCH-less nets in polynomial time. This algorithm, however, cannot be applied to verification of general program nets because Property 1 is an only sufficient condition for general program nets. Let us consider the computation complexity of a problem to verify the termincay for general program nets.

Shepardson et al. have shown that a Turing machine is equivalent to a register machine with the following three instructions [5]:

$I(n)$: Increase the value of register n by 1.

$D(n)$: Decrease the value of register n by 1.

$J(n)[s]$: If the value of register n is equal to 0 then go to statement s.

If a general program net will be able to simulate the above register machine, it has the same modeling power as the register machine. Furthermore, general program nets will be said to have the same modeling power as a Turing machine using Shepardson et al.'s result.

As shown in Fig. 3, program nets can simulate the above three instructions and therefore has the above modeling power as a Turing machine. Since terminacy for Turing machine is undecidable, we give the following theorem for program nets.

Theorem 3: Marking-dependent terminacy problem for general program nets is undecidable. □

5. Example

An execution of this algorithm is shown in Fig. 4.

Figure 4 (a) shows the initial situation of a target program net $(PN_1, \mathbf{d}^0)$.

Figure 4 (b) shows the situation after execution of step 1°. This step deletes node z_{10}, z_{11}, z_{14}, z_{15}, z_{17}, and z_{19} because they are $\mathbf{d}^0$-dead.

Figure 4 (c) shows the situation after execution of step 3°. This step deletes node s, z_1, z_2 because they are reachable from start node s by a path consisting of AND-nodes.

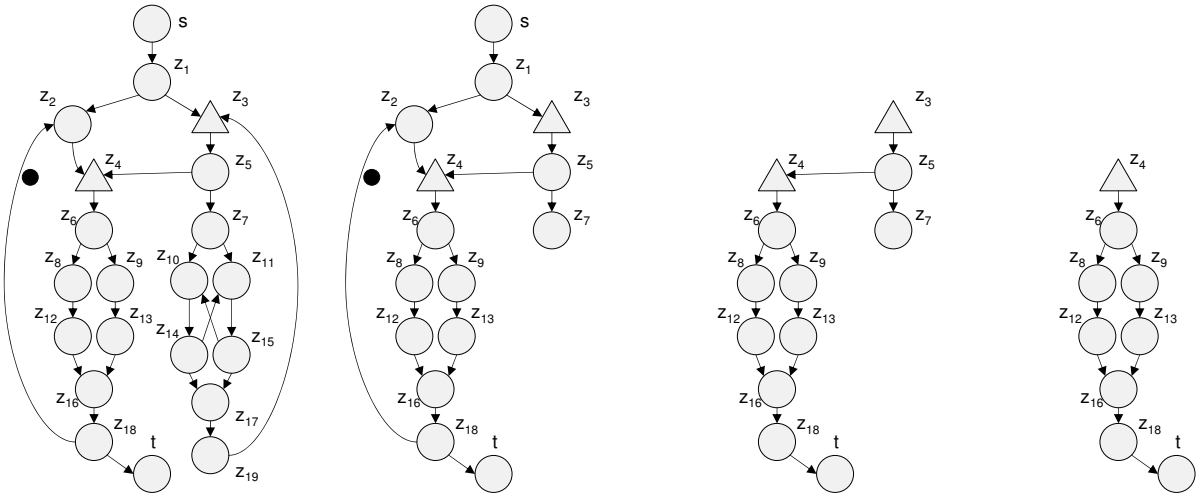
Figure 4 (d) shows the situation after first iteration in execution of loop from 4° to 7°. OR-node z_3 is $\mathbf{d}^0$-Edead. Thus this iteration deletes node z_3, z_5, z_7 because they are reachable from OR-node z_3 by a path consisting of AND-nodes. In the next iteration of this loop, all nodes are reachable from OR-node z_3 by a path consisting of AND-nodes.

There is no node in situation after the iteration. Thus $(PN_1, \mathbf{d}^0)$ can be said to be terminating.

6. Conclusion

In this paper, we have discussed marking-dependent terminacy for program nets. We have first given a necessary and sufficient condition to verify the terminacy. Using this condition, we have constructed a polynomial time algorithm for SWICTH-less nets. We have also shown that the problem to verify the terminacy for general program nets is undecidable.

As a future work, we plan to investigate relationship between the terminacy and dead.



(a) A program net PN_1 at the initial marking $\mathbf{d}^0$.

(b) The situation after execution of step 1°.

(c) The situation after execution of step 3°.

(d) The situation after first iteration in execution of loop from 4° to 7°. On next execution, all nodes are deleted.

Figure 4. Example of execution the algorithm to verify the terminacy for a program net PN_1.

References

[1] K. Onaga, "Data flow Analysis of Program nets," Journal of the Franklin Institute, vol.33, no.5, pp.219–231, 1982.

[2] Q.W. Ge, T. Watanabe, K. Onaga, "Execution Termination and Computation Determinacy of Data-flow Program nets," Journal of the Franklin Institute, vol.328, issue 1, pp.123–141, 1991.

[3] K. Yamada, S. Yamaguchi, Q.W. Ge, M. Tanaka, "Canonical form and Its expression power of Data-flow Program nets," IEICE Technical Report, vol.103, no.537, CST2003-34, pp.5–10, 2003.

[4] S. Yamaguchi, K. Yamada, Q.W. Ge, M. Tanaka, "Dead Problem of Program Nets," IEICE Trans. Fundamentals, vol.E89-A, no.4, pp.887–894, 2006.

[5] J. Shepardson, H. Sturgs, "Computability of Recursive Functions," Journal of the ACM, vol.10, no.2, pp.217–255, 1978.

[6] K. Komiya, S. Yamaguchi, Q.W. Ge, M. Tanaka, "On Verification of Marking-Dependent Terminacy for SWITCH-Less Program Nets," IEICE Technical Report, vol.107, no.472, CST2007-55, pp.53–58, 2008.

Appendix

We give an algorithm to find $\mathbf{d}^0$-dead nodes in SWITCH-less nets as follows.

≪ Find $\mathbf{d}^0$-Dead Nodes ≫

Input : SWITCH-less net $PN_{SWL} = (N, E)$, initial marking $\mathbf{d}^0$

Output : Set of $\mathbf{d}^0$-dead nodes

FIND $\mathbf{d}^0$-DEAD(PN_{SWL}, $\mathbf{d}^0$)

1° $S_{start} \leftarrow \{s\}$.

2° Let $d_{(x,y)}$ be number of tokens on edge (x, y) at $\mathbf{d}^0$. Add the following nodes to S_{start}.

- An AND-node z_{AND} such that $\forall x \in \bullet z_{AND} : d_{(x,z_{AND})} > 0$
- An OR-node z_{OR} such that $\exists x \in \bullet z_{OR} : d_{(x,z_{OR})} > 0$

3° Repeat the following steps for each node $z \in S_{start}$ until $S_{start} = \phi$.

3.1° Execute RECURSIVE-CHECK-FIRABLE(z).

3.2° Delete z from N and S_{start}.

4° Return N.

RECURSIVE-CHECK-FIRABLE(z)

1° Mark z as a visited node.

2° Execute the following steps for each node $x \in z\bullet$, where x is not firable and has not been visited.

2.1° If x is an AND-node and $\forall y \in \bullet x : ((y \in S_{start}) \vee d_{(y,x)} > 0)$ then execute RECURSIVE-CHECK-FIRABLE(x) and delete x from N and S_{start}.

2.2° If x is an OR-node and $\exists y \in \bullet x : ((y \in S_{start}) \vee d_{(y,x)} > 0)$ then execute RECURSIVE-CHECK-FIRABLE(x) and delete x from N and S_{start}.

Computation complexity of this algorithm is $\Sigma_{z \in N} |\bullet z| + \Sigma_{z \in N} |\bullet z|^2 = O(|N| + |E|^2)$[6]. Thus, this algorithm runs in polynomial time.