TM-1042

An Analysis of Parallel Program by Utilizing High-level Nets

by

T. Fukuzawa & H. Hasegawa (Oki)

May, 1991

AN ANALYSIS OF PARALLEL PROGRAM BY UTILIZING HIGH-LEVEL NETS

Toshiyuki Fukuzawa Haruo Hasegawa

Oki Electric Industry Co.,Ltd.
11-22,Shibaura 4-chome,Minato-ku,Tokyo 108,Japan
E-mail: fukuzawa@sot.telcom.oki.co.jp

Abstract *This paper presents procedural analysis methods of a program described in Flat Guarded Horn Clauses(FGHC). FGHC is a parallel programming language based on horn logic. The program processes execute in parallel, so its debug is very difficult. The analysis methods feature that FGHC program is modeled by means of HL-nets(High-Level nets), a kind of Petri net model. The modeling is based on the computation rules of FGHC to analyze behavior of the program. In the modeling rules, "reduction" of FGHC program corresponds to "firing" of a transition. Therefore, the analysis methods calculate firing sequences to detect program deadlock, compute parallelism and evaluate success or failure of the program.*

1 Introduction

Recent advances in computer technology have developed parallel processing computers; accordingly a number of parallel programming languages have been developed in order to utilize the full performance of the computers. The programming languages feature that multiple processes execute in concert with each other, and that execution time can be shorted easily. But, it is difficult to verify programs described in the programming language for the following two reasons:

1. The execution order of processes is not determined, so that bugs may not reappear.

2. Each process executes independently, so that other processes continue executing even if one process stops.

Petri nets are designed to model complex asynchronous and concurrent systems. Petri nets have been applied to verifying programming languages [1][2][3].

HL-nets are a kind of Petri nets model where information is attached to each token. This paper discusses how to apply HL-nets methods in order to verify parallel program described in a logic programming language. The modeling language is FGHC.

2 FGHC

2.1 Format of FGHC

FGHC is a programming language based on horn logic, and added syntactic construct *guard*[4]. An FGHC program is defined as a finite set of guarded horn clauses, in the following format:

$$H:-G_1,\ldots,G_n \mid B_1,\ldots,B_m. \tag{1}$$

Where, H, G_i and B_j are atomic predicates. G_i are restricted to primitive predicates. H, G_i and B_j are called *head goal, guard goal,* and *body goal,* respectively. The operator "|" is called *commit operator. Commit operator* has a role in the control of the program. The left part of *commit operator* is called *guard,* and the right part is called *body.* Also, as a convention, variables are written in capital letters and constants in lower letters.

A goal clause has the following form:

$$H:-B_1,\ldots,B_m. \tag{2}$$

B_j are called *goal.* A goal clause is called empty when m equals 0.

2.2 Procedural Semantics

Informally, an FGHC program execution reduces a given goal clause to the empty clause by means of *input resolution.* This mechanism is called *reduction.* Reduction features two conditions as follows:

1. Unification in guard can not bind caller-side variables. This unification is called *passive unification.*

2. Unification in body may bind caller-side variables. But, only one clause may bind the variables. This unification is called *active unification.*

In passive unification, if a goal attempts to bind a caller-side variable, the goal suspend until another goal binds the variable.

In FGHC, each reduction of goal executes in parallel. The execution is called *AND parallel*. And, if a goal has multiple unifiable clauses, all the clauses are examined in parallel unifiable or not. The execution is called *OR parallel*.

An FGHC program reduces goals recursively. When no reducible clause remains, the program stops. At this time, the program is called *success*, if the goal set is empty. And the program is called *failure*, if the goal set includes any clause(s).

2.3 Interprocess Communication

In FGHC, a goal can be regarded as an independent process. Interprocess communication between the processes is mediated by a common variable as follows: A message-sending process binds a variable. And a message-receiving process reads the variable.

An example of interprocess communication is shown in Figure 1. In this program process $q(X)$ send a message to process $r(X)$.

$$\begin{aligned} p(X) &:- \mathit{true} \mid q(X), r(X). \\ q(X) &:- \mathit{true} \mid X = a. \\ r(X) &:- X = a \mid \mathit{true}. \end{aligned}$$

Figure 1: An example of interprocess communication

2.4 Normal Form

FGHC program clauses can be converted into a format called "normal form" as follows[5]:

$$H :- I_1, \ldots, I_n \mid O_1, \ldots, O_m, B_1, \ldots, B_l. \qquad (3)$$

Here, the components H (head), I_k (input guard), O_j (output guard) and B_i (body) must satisfy the following conditions.

1. All arguments in head must be variables.
2. I_k must be a primitive predicate.
3. O_j must be a primitive predicate binding parameters in head.
4. B_i must be a primitive predicate binding parameters in other bodies, or user-defined predicates.

3 High-Level nets

HL-nets are a kind of Petri nets model where information is attached to each token [6]. The information can be inspected and modified when a transition fires. An enabling condition is represented as an incidence function.

This section defines an incidence matrix of HL-nets without token colors. The matrix has only integer elements. In this paper, the incidence matrix is used to calculate firing sequences.

Definition: Incidence matrix

> The elements of a matrix A^+ are defined by the multiplicity of each input arc. Similarly, a matrix A^- is defined by the multiplicity of each output arc. An incidence matrix A is defined $A = A^+ - A^-$, where row elements represent transitions and column elements represent places. Let n be the number of transitions and m places, and the matrix has dimensions $n \times m$.

4 Modeling Rules

4.1 FGHC Program Limitations

A program subject to the modeling is assumed to satisfy the following constraints:

1. The program is converted into a normal form.
2. No recursive call is allowed.
3. No user-defined predicates is called by different predicates.
4. No more than one predicates binds a common variable.

However, the third constraint is dissolved by changing the symbols of predicate called by multiple clauses. The fourth constraint is dissolved by regarding the program as predicates and removing redundant expressions.

4.2 Modeling in HL-nets

Modeling rules consist of two modeling rules.

4.2.1 The First Modeling Rule

In the first modeling rule, the program clauses parts are converted into HL-nets parts. The correspondences between program parts and HL-nets parts are shown in Table 1.

Table 1: Correspondences between FGHC and HL-nets

FGHC		HL-nets
Clause	→	Transition
Head	→	Input place
Body	→	Output place
Primitive predicate	→	Incidence function
Argument	→	Range of color-function

Clause, head and body Clauses correspond to transitions, heads correspond to input places and bodies composed of user-defined predicates correspond to output places. The correspondences are decided by the computation rules of FGHC. Therefore, the flow of the process controls corresponds to the direction of arc, that is, the reduction in FGHC corresponds to firing in HL-nets.

The two types of parallel execution, *AND parallel* and *OR parallel* are modeled as shown in Figure 2, Figure 3, respectively.

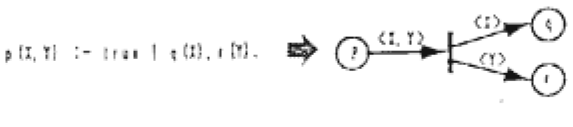

Figure 2: HL-nets graph corresponding to AND parallel

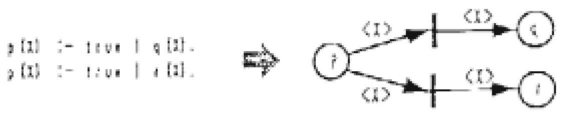

Figure 3: HL-nets graph corresponding to OR parallel

Primitive predicate Primitive predicates don't correspond to transitions or places, but correspond to incidence functions, because the primitive predicates control the flow of the program execution for evaluating true or false.

Argument Arguments correspond to range of color-function. The color expresses the symbol of variables and constants. In particular, goal clause arguments correspond to an initial marking.

4.2.2 The Second Modeling Rule

In the second modeling rule, common variables, functioning as communication channels in interprocess communication, are converted into places. The place is added to the HL-nets generated by the first modeling rule. And two kinds of incidence functions are added. One is a positive incidence function defined on the place and the transition corresponding to the clause which reads the common variable. The other is a negative incidence function defined on the place and the transition corresponding to the clause which binds the common variable.

5 Analysis By HL-nets

5.1 Program success and failure

This section explains how to analyze program success or failure by means of the incidence matrix. The analysis is divided into three steps.

The First Step In the first step are calculated firing sequences of HL-nets modeled by the first modeling rule. However, the firing sequences don't consider colored token. For the purpose of it, solve a matrix equation as follows:

$$M = M_0 + x \cdot CM_1 \tag{4}$$

where CM_1 is an incidence matrix, M_0 is an initial marking which corresponds to the parameters in the goal clause, M is an end marking which corresponds to an end goal of the program, and variable x is a firing vector.

A firing sequence y is derived from the x, if it exists. Existence of a firing vector is a necessary condition for existence of firing sequence. Therefore, none existence of firing sequence satisfying the equation (4) means that the program doesn't reduce to the end goal.

The firing sequences don't consider colored token and asynchronous interprocess communication. They represent all possible reduction sequences in which the program might execute.

The Second Step In the second step are derived firing sequences of HL-nets modeled by the second modeling rule. For the purpose of it, verify that the firing vector x', which has the firing sequence of the HL-nets of the first step, satisfies a matrix equation as follow:

$$M' = M_0' + x' \cdot CM_2 \tag{5}$$

where CM_2 is incidence matrix, M_0' is an initial marking and M' is an end marking.

A firing sequence z is derived from the firing vector x'. The firing sequence z differs from the firing sequence y in that z recognizes the execution order caused by asynchronous interprocess communication.

The Third Step In the third step is derived a complete firing sequence. For the purpose of it, verify that the firing sequence satisfies the conditions for colored tokens to simulate the HL-nets graph. Reduction sequences, which correspond to the firing sequences satisfying the conditions, succeed. The other sequences cause deadlock or failure.

5.2 Deadlock Detection

Deadlocks is detected by firing sequences satisfying the equation (4) and unsatisfying the equation (5). The firing sequences causing deadlock of the HL-nets still causes deadlocks as the reduction sequences for program. Therefore, the causes of the program deadlock are examined by analyzing the causes of firing sequences deadlock.

5.3 Computing Parallelism

A parallelism in the FGHC program can be defined by a firing vector and a firing sequence as follows:

$$\mathit{Parallelism} = \frac{\mathit{Sum\ of\ firings}}{\mathit{Length\ of\ a\ firing\ sequence}} \tag{6}$$

$$\mathit{Max\ parallelism} = \mathit{Maximum\ number\ of\ CET}, \tag{7}$$

$$\mathit{where}\ \ \mathit{CET} = \mathit{concurrent\ enabled\ transitions}$$

6 Example

In this section is explained analysis of an example. A program and the HL-nets graph are shown in Figure 4. And the incidence matrix is shown in Table 2.

```
p (X, Y)  :-  true | q (X), r (Y), s (X, Y).   %1
q (X)     :-  true | X=a.                      %2
q (X)     :-  true | X=b.                      %3
r (X)     :-  true | X=c.                      %4
r (X)     :-  true | true.                     %5
s (X, Y)  :-  X=a, Y=c  | true.                %6
s (X, Y)  :-  X=b, Y=d  | true.                %7

?- p (X, Y).
```

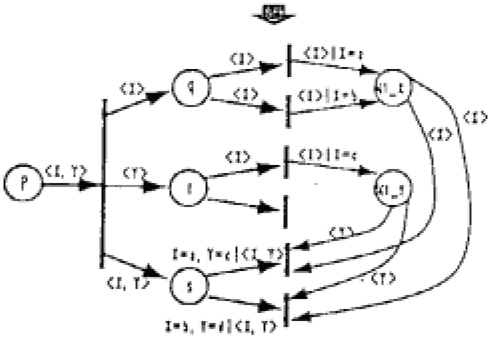

Figure 4: A program and the HL-nets graph

Table 2: The incidence matrix of the HL-nets model

-	p	q	r	s	%1_X	%1_Y
%1	-1	1	1	1	0	0
%2	0	-1	0	0	1	0
%3	0	-1	0	0	1	0
%4	0	0	-1	0	0	1
%5	0	0	-1	0	0	0
%6	0	0	0	-1	-1	-1
%7	0	0	0	-1	-1	-1

First, solve the equation (4). The initial marking $M_0 = (1,0,0,0)$, an end marking is $M = (0,0,0,0)$. Then, the following 8 vectors are solutions of the equation. Each vevtor generates a firing sequence.

$a_1 = (1,1,0,1,0,1,0), a_2 = (1,1,0,1,0,0,1),$
$a_3 = (1,1,0,0,1,1,0), a_4 = (1,1,0,0,1,0,1),$
$a_5 = (1,0,1,1,0,1,0), a_6 = (1,0,1,1,0,0,1),$
$a_7 = (1,0,1,0,1,1,0), a_8 = (1,0,1,0,1,0,1)$

Next, verify that each vector satisfies the equation (5). The HL-nets model is added to two places corresponding to communication channels. The initial marking $M_0' = (1,0,0,0,0,0)$, the end marking $M' = (0,0,0,0,0,0)$. Then, the following four solutions, a_1, a_2, a_5 and a_6, satisfy the equation.

Finally, verify each sequence satisfies the conditions of the colored tokens. Let $x = (\%1, \%2, \%3, \%4, \%5, \%6, \%7)$, where $\%i$ means that transition number. The firing sequences corresponding to these solutions are the following. Note that clauses in parentheses () have no sequential relationship.

$b_1 = \{\%1 \rightarrow (\%2, \%4) \rightarrow \%6\},$
$b_2 = \{\%1 \rightarrow (\%2, \%4) \rightarrow \%7\},$
$b_5 = \{\%1 \rightarrow (\%3, \%4) \rightarrow \%6\},$
$b_6 = \{\%1 \rightarrow (\%3, \%4) \rightarrow \%7\}.$

The only one sequence b_1 satisfies the conditions of colored token. Sequences b_2, b_5 and b_6 are unable to fire the last transition %6 or %7. And, the firing sequences b_3, b_4, b_7 and b_8 cause deadlock.

The following conclusion is derived. This program is capable of eight different execution sequences. However, the only one sequence b_1 succeeds. Sequences b_2, b_5, and b_6 fail, because a transition do not fire. Sequences b_3, b_4, b_7 and b_8 cause deadlock because a common variable %_Y is not bound. The number of firings equals 4 and the length of the firing sequence equals 3. Thus, the parallelism equals $1.3 = 4/3$ and maximum parallelism equals 2.

7 Conclusion

This paper has discussed the rule for modeling a program described in FGHC on HL-nets, and procedural analysis methods of the program, The modeling features that the FGHC primitive predicates correspond to HL-nets incidence functions, and controls over asynchronous interprocess communication are represented. Future topics of research will include the expansion of program applications. such as stream communication, recursion, and large scale program.

Acknowledgements This work was researched as a part of the Fifth Generation Computer Systems Project in Japan. We would like to thank our project members for very useful discussions. We also would like to express our gratitude to Dr.Ryuzo Hasegawa, Deputy Manager of ICOT Research Department, for giving us many helpful suggestions.

References

[1] H.Hasegawa et al.:*An Analysis of Parallel Logic Program by utilyzing Petri Nets*, IEICE, CAS Karuizawa Workshop, pp.199-206, 1990.(In Japanese)

[2] S.M.Shatz,W.K.Cheng: *A Petri Framework for Automated Static Analysis of Ada tasking Behavior*, Journal of Systems and Software, 8, pp.343-359, 1988.

[3] J.Jeffrey,T.Murata: *A High-Level Petri Net for a Subset of FGHC*, SEKE'90, pp.260-266, 1990.

[4] K.Ueda: *Garded Horn Clauses*, Tech. Report TR-103, ICOT, 1985.

[5] K.Ueda,K.Furukawa: *Transformation Rules for GHC programs*, Proc. of FGCS'88, pp.582-591, ICOT, 1988.

[6] G.Goos,J.Hartmanis: *Advances in Petri Nets, PartI*, Lecture Notes in Computer Science, pp.207-237