Title	Transformations of Communicating Sequential Processes(Mathematical Theories on Computing Schemes and Their Applications)
Author(s)	Musha, Hiroyuki; Tokuda, Takehiro
Citation	数理解析研究所講究録 (1983), 494: 126-136
Issue Date	1983-06
URL	http://hdl.handle.net/2433/103571
Right	
Type	Departmental Bulletin Paper
Textversion	publisher

数理解析研究所講究録
494巻 1983年 126-136

Transformations of Communicating Sequential Processes

Hiroyuki Musha and Takehiro Tokuda
武舎 広幸　　徳田 雄洋
Department of Computer Science
Yamanashi University

1. Introduction

Distributed computing models are natural and powerful systems for describing both concurrent and sequential computing phenomena and they gain growing interests in connection with VLSI technology. The system of Communicating Sequential Processes (CSP) is one of those models proposed by Hoare in [1], where input and output commands of processes are considered basic primitives. Transformation of CSP into sequentially executable programs is discussed in this paper.

The authors are working on a project which we call CSP Programming Support Environment Project [2], where CSP is considered one of the tools for describing algorithms. The goal of the project is to construct a number of automatic (or semi-automatic) transformation systems among different types of computatoion models including, for example, attribute grammar systems, Prolog-type computing models and distributed computing models. As for CSP model, a transformer which transforms algorithms expressed in CSP (CSP programs for short) into sequentially executable programs is being designed. For the first stage, coroutines are considered the initial target of the transformation.

The rest of this paper is organized as follows. In the next

section definitions of CSP and coroutines are given. Section 3 states three kinds of algorithms for transformation, and the last section presents conclusion.

2. Preliminaries

2.1 Definitions of CSP

In this section, sysntax and semantics of CSP are informally described. A CSP program is a collection P of processes, which share no common variables at all and are supposed to be executed concurrently. Communication between two processes p,q P is expressed by the input and output commands "q?v" and "p!e", where e is an expression and v is a variable in which the received value of the expression of e is assigned. Execution of these commands are syncronous, i.e. p waits at "q?v" until q is ready to output the message at "p!e" and vice versa.

Constituents of each process are commands based on Dijkstra's guarded commands, which can be classified into two types: simple commands and structured commands. The members of the simple commands are the assignment command, the input command, the output command and the null command which does nothing and is denoted by "skip". Structured commands, alternative and repetitive commands, are organized by a set of guarded commands and express selective and repetitive execution.

A guarded command is executed when its guard does not fail. An alternative command fails if all guards fail. A repetitive command specifies as many iterations as possible of its constituent alternative command, and it terminates when all guards fail. The following examples are adopted from [1].

Examples

(1) [x >= y --> m := x [] y >= x --> m := y]

Here the value of m bocomes MAX(x,y).

(2) i := 0; *[i<size; content(i)<>m --> i := i+1]

This program finds an element which is equal to n of the array "content".

An input command can appear in the end of a guard and is executed only when a corresponding output command is executed. The input command fails if the process specified is terminated; the execution suspends if the corresponding process is not ready to output, which can result in deadlock.

The following example is a CSP program which soleves the famous 8-Queens Problem [3].

```
[TRY(i:1..8)::
   A:(1..8) boolean; B:(2..16) boolean;
   C:(-7..7) boolean; X:(1..8) integer;
   *[TRY(i-1)?(A,B,C,X) -->
      j:integer; j:=1;
      *[j<=8; A(j); B(i+j); C(i-j) -->
         X(i) :=j;
         A(j)   := false;
         B(i+j) := false;
         C(i-j) := false;
         TRY(i+1)!(A,B,C,X);
         A(j)   := true;
         B(i+j) := true;
         C(i-j) := true;
         j := j+1       ]]
||
TRY(0)::
   A:(1..8) boolean; B:(2..16) boolean;
   C:(-7..7) boolean; X:(1..8) integer;
   i:integer;
   i:=1; *[i<=8  --> A(i):=true; i:=i+1];
   i:=2; *[i<=16 --> B(i):=true; i:=i+1];
   i:=-7;*[i<=7  --> C(i):=true; i:=i+1];
   i:=1; *[i<=8  --> X(i):=0;    i:=i+1];
   TRY(1)!(A,B,C,X)
||
TRY(9)::
   A:(1..8) boolean; B:(2..16) boolean;
   C:(-7..7) boolean; X:(1..8) integer;
   *[TRY(8)?(A,B,C,X) -->
      PRINT!X          ]
]
```

Program 1. 8-Queens Problem in CSP.

2.2 Definitions of coroutines

A set of coroutines is the target of our transformation. A coroutine is defined as a routine (subprogram) which has the following two features:

(1) the values of variables local to the routine are retained between successive activations of the routine, and

(2) when the control reenters the routine, the execution resumes at the point where it left off last time.

3. Transformation

Algorithms of transformations are described informally in this section. A detailed discription of the algorithms is in [4]. In 3.1, we present a general method of transformation which can be applied to any kind of CSP programs. 3.2 describes a somewhat optimized algorithm which is effective when a CSP program contains no input guard. In 3.3, more efficient algorithm which can be applied to a set of CSP programs which satisfies some conditions concerning about the forms of communication among processes.

Generally, a process of a CSP program will be transformed into a coroutine which strictly corresponds to the original process, each command of the process, except for commands containing input and output, will be changed into the same command of the target; and a schduler of those coroutines is introduced to manage the selection and excution of those coroutines. Communication among processes are realized by introducing global variables in which values of the messages, types of the messages and statuses of the routines will be stored.

3.1 Algorithm for Transformation (1)

An algorithm for transformation which can be applied to general CSP programs is presented in this subsection.

As stated at the top of this section, every process of a CSP program is transformed into a coroutine and each command of each process is transformed into the same command except for input and output commands. A special routine called scheduler is added to those coroutines and it always keeps the statuses of every process and control the execution.

The scheduler repeats the following until no process is executable:

(1) The scheduler chooses and activates a process which is executable;

(2) The chosen process executes its commands as far as it can proceed (The process can not proceed when it can not send or receive a message at an input or an output command, or when it is terminated.);

(3) The suspended process returns the control to the scheduler.

The mechanism of the message passing is as follows. In principle the process which reaches the place of rendezvous first writes the type of the message (and the value of the message if the process is the sender of the message) in globally accessible space, then the other process which reached the input or output command checks the correspondence of the message, and at last the value of the message is assigned to the target variable of the input process.

A somewhat special treatment is necessary for input guards. When the control reached an alternative or a repetitive command with input guards, the guards of the command whose results can be

determined at that moment are evaluated. If there is any guard which does not fail, the guard is selected for execution. If such a guard does not exist, the execution of the process is suspended and it returns the control to the scheduler. Then, the control flows on other processes in the way stated above, until success of one of the input guards is informed by the sender of the message or failure of all the guards are settled by termination of all the processes related.

3.2 Algorithm for Transformation (2)

If the CSP program to be transformed has no input guard, a strategy called demand driven reduce the duty of the scheduler, since in those cases the participants of a paticular communicatin is always determined uniquely.

If a process p reaches an input or an output command, the process transfer the control to (or call) the partner of the rendezvous. The activated (called) process proceeds execution, until it reaches the place of the rendezvous and answer the request of the activator. It is possible that the called process also activates other processes, however, those called processes can not call any process which is already waiting for the partner to respond the request, since this indicates the presence of deadlock. Thus, if there does not arise any deadlock, the control must return to the first process p. In this way the execution proceeds until the process p chosen by the scheduler terminates and return the control to the scheduler. Thus, the number of the scheduler's choice of the process to activate decreases comparing with the first algorithm.

3.3 Algorithm for Transformation (3)

If a CSP program satisfies certain conditions stated in this subsection, the role of the scheduler is to only activate a special process called source process which leads the computation. In this subsection we first states the conditions to be satisfied and, then, describe the way of the transformation.

Any program in CSP that satisfies three conditions stated below in terms of communication graph [5] and activation graph can be executed without arbitrary choice of the process by the scheduler. At first the communication graph Gc is defined as follows:

Definition 1 (communication graph Gc).

For a given CSP program, the COMMUNICATION GRAPH Gc is defined as follows:

(1) each process is a point of Gc, and

(2) if there is a communication (i.e. transfer of messages) from a process p to a prcess q, pq is contained in Gc as an arc.

Let P be the set of processes of the CSP program, or, in other words, the set of points of Gc, and let Ac be the set of arcs in Gc. We write as Gc=(P,Ac). The first condition to be satisfied is as follows:

Condition 1.

Gc is acyclic.

By this condition a partial order $\preceq$ can be naturally induced into the set of points P of Gc. We shall define the order of $p,q \in P$ as

$pq \in Ac \quad ==> \quad p \preceq q.$

Suppose that an expression in CSP satisfies the above condition. The next condition we take into consideration is:

Condition 2.

There exists a process $s \in P$ such that for any $p \in P$, $p \preceq s$ or $s \preceq p$ holds.

The process which satisfies the above condition may not be unique. If there exists more than one process, we select an arbitrary one and fix it from now on.

Definition 2 (source process s).

We select a process which satisfies Conditin 2 and fix it. We call the process the SOURCE (PROCESS) of P, which will be denoted by s.

Another kind of graph called activation graph is now defined.

Definition 3 (activation graph Ga)

If Condition 1 and 2 are satisfied, the ACTIVATION GRAPH $Ga=(Pa,Aa)$ of a CSP program can be defined as follows:

(1) $Pa=P$ (the set of points is the same as that of Gc; thus we use P instead of Pa), and

(2) $pq \in Aa \quad <==> \quad qp \in Ac$ (if $q \preceq s$)

$pq \in Ac$ (if $s \preceq p$).

Note that indegree of the graph Ga of the source process s is $\emptyset$. The following proposition is proved easily.

Proposition 1

Ga is acyclic.

By this proposition, a new partial order $(P,\overset{a}{\leq})$ which is different from $(P,\leq)$ is induced; it represents the order of the activation.

The last condition to be satisfied is stated in terms of the activation graph Ga as follows:

Condition 3.

Ga is an out-tree [6].

Remark

The above condition is equivalent to the following one:

Condition 3'.

With respect to Ga, the indegree of the source s is $\emptyset$ and the indegrees of all other processes are 1.

For convenience, we will define several terms.

Definition 4 (producer and consumer).

A process p is said to be a PRODUCER (PROCESS) if $p \not< s$, and is said to be a CONSUMER (PROCESS) if $s \not< p$.

Definition 5 (parent, son, ancestry and descendant).

For each $p \in P$, we define the following:

(1) if $qp \in Aa$, q is said to be the parent of p (which is uniquely defined),

(2) if $pq \in Aa$, q is said to be a son of p,

(3) $\{q \in P \mid q \not< p\}$ is said to be ancestry of p, and

(4) $\{q \in P \mid p \not< q\}$ is said to be descendant of p.

Those programs which satisfies above three conditions can be executed in the following manner.

(1) The shceduler activates the source process s.

(2) The source process makes all the producer processes be ready to send messages in the following way:

(2-1) it activates each of its son that is a consumer, and then

(2-2) each of those sons also makes their sons be ready to send messages by activating them.

(2-2) proceeds until all the producers are activated and becomes ready to send messages to their parent processes.

(3) The computation proceeds by the leading of the source process preserving the condition that all the producer processes are always be ready to send messages unless they are terminated. (Note that the result of the input guards can be always determined since all the producers are ready to send messages.)

(4) When the source process is reached its end, it broadcasts its termination to all the consumers in the same manner as (2-2).

(5) All the consumers change their statuses to "terminated" and return the control to their parents.

(6) The source process gets the control again and returns it to the scheduler.

(7) The execution terminates.

4. Conclusion

Three algorithms for transformatin of CSP programs into sequentally executable forms are presented. Those algorithms must contribute to the problem of scheduling of processes of CSP programs.

References

[1] C.A.R. Hoare: Communicating Sequential Processes, Comm. ACM, Vol. 21, No. 8 (Aug. 1978), pp. 666-677.

[2] T. Tokuda: CSP Programming Support Environment Project, Proc. of the 26th Conference of IPSJ, to appear.

[3] M. Sassa et al.: Programming with Streams, Proc. of the 25th Conference of IPSJ, pp. 257-258, 1982.

[4] H. Musha: On Effective Transformatins of Communicating Sequential Processes, Master Thesis, Yamanashi University, 1983.

[5] N. Francez: Distributed Termination, ACM Trans. on Prog. Lang. and Sys., Vol. 2, No. 1 (Jan. 1980), pp. 42-55.

[6] F. Harary: Graph Theory, Addison-Wesley, 1969.